uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Telemática
2016-2017

Trabajo Fin de Grado

Evaluación de prestaciones de páginas web sobre QUIC

Laura Balart Sánchez-Ortiz

Tutor: Carlos García Rubio

Leganés, 16 de octubre de 2017

Agradecimientos

En primer lugar, me gustaría agradecer a mi familia, por apoyarme durante estos cuatro años de carrera, por calmarme y animarme a seguir en todos los momentos duros, exámenes suspensos, prácticas interminables y, por último, con este proyecto, que bien saben lo necesario que ha sido ese apoyo.

A Ana, por vivir todo este proceso desde aquella primera clase de cálculo I hasta hoy, que por fin conseguiremos nuestra meta de ser ingenieras. Han sido cuatro años de retos, de superarnos a nosotras mismas y de reír hasta que se nos escapaban las lágrimas ante cada obstáculo. Sin ti, este proceso hubiera sido infinitamente más duro. Muchísimas gracias, leona.

A BEST, esa asociación de estudiantes que me ha acompañado durante todos estos años. Con cada uno de los miembros que habéis formado parte de la asociación estos cuatro años he aprendido a salir de mi zona de confort, a dar todo lo posible de mi misma para sacar nuestros proyectos adelante, y a conseguir la mejor versión de mi misma.

A Aitor, que, aunque se haya unido tarde a este reto, ha sido uno de mis pilares principales para cada momento duro, error o tarde interminable frente al ordenador. Has hecho estos últimos meses de carrera mucho más llevaderos. Te quiero.

Al "TFG Team", Ana, Luisi, George y Ionut, por hacer la piña que hemos hecho, hacer fluir la información entre todos en cuanto le llegaba a uno y animarnos mutuamente durante todos estos meses.

Por último, me gustaría agradecer a mi tutor, Carlos García Rubio, por motivarme e interesarme tanto en los protocolos de la capa de aplicación con su asignatura "Aplicaciones Telemáticas", así como por proponerme un tema tan interesante para este Trabajo de Fin de Grado.

"No pases por la Universidad, haz que la Universidad pase por ti"

Executive Summary

Human beings have always tried to improve themselves, and in the latest years, innovation has been one of our main concerns and points of effort.

In the 1950s, the first electronic computers were being developed, and the first drafts of the idea to connect them and make a network of computers started to arise [1]. It was not until 1960s when Robert Taylor and Lawrence Roberts started developing ARPANET (Advanced Research Projects Agency Network), a packet switching network that was meant to be the first one implementing TCP/IP protocol suite during the 1970s. In 1969, the first message was sent over this network [2].

During all these forty years, we have managed to improve and promote that early idea of the Internet to the point that now, we can't even imagine how our lives would be without it. New solutions and ideas on this field are being proposed day after day to keep evolving, to strive for the best.

This willing to improve internet as we know it today, is one of the reasons why QUIC has been under development for 5 years now and strives to be the next transport layer protocol for web retrieving, intending to eliminate the use of HTTP/2 over TCP + TLS so that web pages use HTTP/2 over QUIC [3].

On the one hand, HTTP/2 implements the following features as defined in its RFC [4]:

- Its basic unit is the *frame*, which is 24 bits long.
- Since HTTP/2 usually runs over TCP and TLS, to establish a connection it must go through:
    - TCP Three-way handshake: it consists on the client sending a SYN frame, then the server will answer with a SYN-ACK frame and the client will answer again with a ACK frame.
    - TLS v1.2 connection establishment:
        - The client will send a *Client Hello* packet and the server will answer with a frame containing the *Server Hello*, the certificate, the server key exchange, a request for a certificate from the client and a message indicating that the Server Hello finishes called *Server Hello Done*.
        - Then, the client will send a frame containing the certificate (if requested), the client key exchange, the results of the verification of the certificate sent by the server and use the protocol Change Cipher Spec to change to the cipher they specified in the previous

exchange. The frame will finish with a *Finished* message indicating that the frame is ended.
        - Finally, the server will use the Change Cipher Spec to change its communication and finish with the *Finished* message.
    - As we can see, the use of both protocols means that it will be needed from 1 to 3 round trip times (exchanges with the server) before the connection is established.
- It uses one only TCP connection: Since the previous HTTP versions used one connection for each resource, it could lead to a blocking of the resources being transmitted if one of the resources took too long to transmit or got lost. Since HTTP/2 implements multiplexing, the server can answer request in a different order of arrival over one unique TCP connection.
- It is a binary protocol: which means that it is more compact and less susceptible to errors.
- It allows prioritization of streams and flow control. That means that, since some elements of a web page are more important than others, the server will be able to answer the request in order of importance to retrieve those elements as fastest as possible. HTTP/2 uses the flow control to ensure that the different streams of information over a single TCP connection doesn't interfere destructively with one another.
- It uses a mechanism for header compression known as HPACK, which provides security for headers transmission and avoids redundancy since the headers sent to the same server are usually quite similar.
- Finally, it introduces a service known as "server push", that allows the server to send information to the client without having previously received an explicit request for those resources.

On the other hand, QUIC (Quick UDP Internet Connection) is a multiplexed transport protocol that runs over UDP and intends to be 0-RTT by replacing TCP and TLS - the protocols that HTTP/2 is using - with UDP as transport protocol and its own security encryption (QUIC-Crypto). Right now, it is available as an experimental feature and, if you're a user of Chrome Web Browser, it is highly possible that you are using it without noticing it [3].

Some features that QUIC provide, as stated in the IETF entry for its draft [5] include:

- Multiplexing: Just as HTTP/2, QUIC allows different and simultaneous streams of information, but, since it runs over UDP and not over TCP, it doesn't require in-order delivery of the packets, so in case a packet gets lost, the rest of the packets can be delivered without problems.

- Connection establishment: QUIC aims to have a 0-round trip time connection establishment. This is achieved by saving in the cache memory the credentials used for the connection the first time the client connects to a specific server. The following times QUIC will retrieve those credentials from the cache memory, being able to send information in the first exchange of packets.
- QUIC-Crypto: QUIC uses its own encryption algorithm that decrypts packets independently of the rest of the packets to avoid the need for in-order delivery that it managed to skip with the use of UDP.
- Forward Error Connection (FEC): This allows QUIC to recover the information of a lost packet by using the FEC packet sent at the beginning and the rest of the packets of that group, without needing a retransmission of the lost packets. This is especially useful in network scenarios with high round-trip times, since a retransmission of a packet would take more time to be retransmitted, and thus, affect more to the web page load time than in a connection with low round trip times.
- Flexible congestion control: QUIC implements different mechanisms for congestion control that allows them to differ between the ACKs that indicate that a new packet must be transmitted or that it is needed a retransmission of a previous packet.
- Connection flow control: It follows HTTP/2 flow control, and implements flow control to a connection level which limits the buffer that a QUIC receiver can assign to a connection. This buffer applies to all the streams in the connection and overrides its flow control if needed.
- Prevention against header manipulation and injection attacks: Since HTTP headers are transmitted in clear and are not authenticated, that could lead to some injection and manipulation of headers attacks. Since QUIC packets are always authenticated and its information is usually encrypted and since the headers must be also authenticated by the receptor, any attack of header manipulation would be frustrated.
- Connection migration: Since TCP connections are identified by the IP address and port number of the destination device and IP address and port number of the origin device, the connections don't survive to changes in the port numbers or IP addresses (for example, when changing from a Wi-Fi connection to a cellular one). QUIC identifies its connections by a connection ID, so it can survive changes in any of those fields without compromising the connection.

To use HTTP/2 over QUIC, the server has to announce that it understands QUIC as an alternative protocol by the use of `Alternate-Protocol: 443:quic` header (it can

listen to QUIC in any port, but Google Chrome – the only browser that is implementing its use in the moment – only allows the use of QUIC on secure connections).

With all these features, QUIC seems to be a promising protocol designed to fit HTTP/2 features, but will HTTP/2 over QUIC improve the performance of HTTP/2 over TCP and TLS?

That is the main objective for this study: compare the load time of webpages using HTTP/2 over TCP and TLS and over QUIC under different network conditions and testing them against webpages of different characteristics (weight, number of request needed to retrieve the full page….)

Since QUIC is not an official protocol yet, right now the only servers that understand QUIC are the ones that come from Google Inc. that include the header `Alternate-Protocol: 443:quic` in their HTTP responses, which limits the number of pages that we can test. We have chosen fifty-three web pages belonging to Google Inc. to compare both protocols. The list of these pages can be found in the Appendix 1.

About the network configurations, our main objective was that the results from this study were applicable to the real world, so we chose to try it by combining high and low download and upload speed, high and low latency and four different percentages of packet losses.

So, as a summary, we will perform ten page loads of each web page, under each of the sixteen network configurations with both protocols, which makes a total of 16.960 page loads.

To automate the process, we have used two different scripts:

1. To load the webpage and record its load time, we have modified a script [6] made by Somak R. Das for his study “Evaluation of QUIC on Web Page Performance” [7], that calculates the time difference between the moment when the user perceives that the page starts to load and the time instant when all the resources have been loaded. It saves the web page and its load time in a text file.
2. To emulate the network and repeat the page loads ten times over the different network configurations, we created a script that used Mahimahi [8] to emulate the network and created different loops so that it uses the previous script several times to get all the page loads needed.

The results from all these webpage loads can be found in the 9. Results section, and the scripts used, in Appendix IV and V at the end of this document.

To process the results, we have calculated the arithmetic average of the ten loads for each web page, the variance between the ten samples and the confidence interval to ensure that the measurements were acceptable for this study.

Once these results have been processed, we can conclude:

- In networks with high upload and download speeds, QUIC proves to be more efficient than HTTP/2 over TCP and TLS, independently on the webpage characteristics.
- In networks with low bandwidth, there is no significant difference between the two different protocols.
- In general, QUIC performs better on heavy web pages.
- The Forward Error Correction that QUIC implements works well under the 0.1% and 1% lossy network scenarios, making QUIC outperform TCP and TLS under these circumstances. In the cases of 10% of losses, TCP and TLS usually perform better than QUIC.
- An increase on the latency benefits QUIC, increasing its performance.
- In networks with high bandwidth and high latency, QUIC outperforms TCP and TLS on pages that need a lot of requests, while in networks of low bandwidth and latency, the combination of TCP and TLS performs better on the same pages.

As a conclusion, QUIC is obtaining good results under many of the network conditions, but there is still a lot of development ahead to achieve their goals and manage to substitute TCP and TLS as the main webpage retriever transport protocol.

Índice de contenidos

1. Índice de gráficos

2. Índice de figuras

3. Lista de acrónimos

- ACK: Acknowledgement
- ANR: Autoridad Nacional de Reglamentación
- ARPANET: Advanced Research Projects Agency Network
- CA: Certificate Authority
- CHLO: Client Hello
- CNMC: Comisión Nacional de los Mercados y la Competencia.
- FEC: Forward Error Correction
- HOL: Head Of Line
- HTTP: Hypertext Transfer Protocol
- IESG: Internet Engineering Steering Group
- IETF: Internet Engineering Task Force
- Kb: Kilo bits
- KB: Kilo Bytes
- MB : Mega Bytes
- Mb: Mega bits
- Ms: milisegundos
- MTU: Maximum Transmission Unit
- QUIC: Quick UDP Internet Connection
- REJ: Rejection
- RFC: Request For Comments
- RTO: Recovery Time Objective
- RTT: Round Trip Time
- RTTVAR: Round Trip Time Variation
- SHLO: Server Hello
- SRTT: Smooth Round Trip Time
- SSL: Secure Sockets Layer
- SYN: Synchronize
- TCP: Transmission Control Protocol
- TLP: Tail Loss Probes
- TLS: Transport Layer Security
- UDP: User Datagram Protocol
- URL: Uniform Resource Locator

4. Introducción

4.1. Estado del arte y motivación

Desde la aparición de Internet, siempre se ha intentado mejorar la facilidad de acceso de las páginas web para los usuarios: su tiempo de carga, accesibilidad, seguridad…

Actualmente, el protocolo HTTP es, de lejos, el más utilizado en cuanto a protocolos de la capa de aplicación para la recuperación de páginas web y durante los últimos años se han trabajado en mejoras sobre este protocolo para mejorar su rendimiento [9].

Estas mejoras incluyen múltiples conexiones TCP simultáneas, arquitectura *pipeline,* HTTP persistente multiplexado a través de un número reducido de conexiones TCP, multiplexación de flujos paralelos de HTTP sobre una única conexión TCP, priorización de solicitudes y respuestas, compresión de encabezados HTTP y servidores "*push*" [10].

Como hemos dicho, estas mejoras se han producido sobre el protocolo HTTP, llegando a su última versión oficial: HTTP/2. Aun así, Google ha estado desarrollando desde el año 2012 un protocolo de transporte que pretende desafiar a HTTP/2, llamado QUIC.

QUIC (Quick UDP Internet Connection) es un protocolo experimental de multiplexación sobre UDP, que pretende ser 0-RTT reemplazando TCP y TLS con UDP y su propia seguridad. Actualmente, se encuentra disponible de manera experimental en el navegador Chrome y algunas de nuestras conexiones del día a día se realizan utilizando este protocolo sin que nos demos cuenta [3].

4.2. Objetivos

Debido a que QUIC es un protocolo experimental, se han realizado muchas mejoras y cambios en estos cinco años de desarrollo, mejorando la accesibilidad, tiempo de carga, y rendimiento del protocolo.

En este estudio, nuestra meta será comparar el uso de HTTP/2 sobre la combinación de TCP y TLS, utilizado por la mayoría de servidores hoy en día, con su potencial adversario QUIC.

Esto no será una tarea fácil puesto que, al ser QUIC un protocolo en desarrollo, no está disponible en la mayoría de servidores, como TCP y TLS, sino que los únicos servidores que entienden este protocolo son los servidores de Google Inc., que incluyen la cabecera `Alternate-Protocol: 443:quic` en sus respuestas HTTP.

Además, las páginas web tienen una gran variedad en los componentes que las forman, y los usuarios acceden a la red a través de distintos medios de red (redes móviles, redes inalámbricas y redes por cable).

El objetivo de este documento será evaluar las páginas que actualmente están accesibles para el protocolo QUIC, para después comparar su rendimiento con el de HTTP/2 sobre TCP y TLS en esas mismas páginas bajo distintas condiciones de red.

4.3. Marco regulador

4.3.1. IETF

El mundo de los protocolos de internet actualmente se encuentra regulado por el *Internet Engineering Task Force (IETF)*, o Grupo de Trabajo de Ingeniería de Internet en español, una organización internacionalidad de normalización cuyo objetivo es regular las propuestas y estándares de internet (RFC) [11].

La misión de IETF es "hacer que el Internet funcione mejor, produciendo documentos técnicos relevantes y de alta calidad que influyan en la manera de diseñar, usar y administrar Internet" [12].

Actualmente, cualquier protocolo que quiera convertirse en un estándar tiene que pasar por las siguientes fases [13]:

1. Publicación como *Internet-Draft*
2. Cuando se da una recomendación por el responsable del grupo de trabajo del IETF a su director de área, se inicia una acción de estándares.
3. Después de que el documento sea corregido y revisado múltiples veces, si cumple los criterios y tiene la calidad y claridad técnica necesaria, el IESG (Internet Engineering Steering Group) aprobará la acción de estándares.
4. Una vez esté aprobada, el editor de RFCs se encargará de publicar el estándar definitivo.

De los protocolos que hablamos en este documento, UDP, TCP, TLS, HTTP/1.1 y HTTP/2 están aceptados actualmente como estándar, mientras que QUIC se encuentra en la fase de *Internet-Draft*, aunque ya se ha iniciado la acción de estándares y ha sido revisado múltiples veces.

4.3.2 Regulación acerca del acceso a internet.

El acceso a internet está definido en el Marco Regulador de las Comunicaciones Electrónicas [14], una Directiva europea en el que se da la competencia de regular el acceso a internet a las ANR (Autoridades Nacionales de Reglamentación).

En específico, la Autoridad Nacional de Reglamentación española es la Comisión Nacional de los Mercados y la Competencia (CNMC) [15].

La CNMC se encarga de analizar el mercado para observar el nivel de competencia e imponer obligaciones para regularlo, resolver los conflictos entre operadores o con otras

entidades, establecen las condiciones técnicas para la portabilidad, regulan el coste universal y sancionan a las entidades que no cumplan esas normas [16].

Debido a que no existe ninguna regulación acerca de los protocolos que deben ser usados para la carga de páginas web, sino simplemente del acceso a internet por parte de las operadoras, no profundizaremos más en esta entidad reguladora.

4.4 Entorno socio-económico

4.4.1 Presupuesto

NOMBRE	PRECIO TOTAL
SISTEMAS OPERATIVOS	
Windows 10 Home	135€
Ubuntu Desktop 16.04	0€
PROGRAMAS	
Microsoft Office 365 (7€ al mes durante 9 meses)	63€
Python	0€
Chrome WebDriver	0€
Selenium	0€
Mahimahi	0€
Wireshark	0€
Google Chrome	0€
MATERIAL	
Ordenador portátil ACER Aspire E5-573-56XH	529€
SERVICIOS	
Conexión a Internet (20€ al mes durante 9 meses)	180€
Mano de obra (ingeniero) (2.000€ brutos al mes durante 9 meses)	18.000€
TOTAL	**18.907€**

Tabla 1 - Presupuesto del proyecto

Para el presupuesto hemos tenido en cuenta el precio de las licencias utilizadas durante un año, tanto para sistemas operativos como programas, para la parte técnica como para la redacción de este documento. Por otro lado, también hemos tenido en cuenta el material utilizado (ordenador portátil) y los servicios contratados (conexión a internet), así como un ingeniero que se encargue de realizar las pruebas. Para el coste del ingeniero hemos estimado 2000€ brutos al mes, y para la duración del proyecto nos hemos basado en el diagrama de Gantt que se puede encontrar en el Anexo II (duración total del proyecto: 39 semanas o 9 meses).

4.4.2 Impacto socio-económico

El tiempo de carga de las páginas web ha sido algo crucial durante los últimos años, debido a que a un usuario de internet no le gusta tener que esperar durante mucho tiempo para poder cargar los contenidos que desea ver.

Debido a que en este estudio comprobamos qué protocolo actúa mejor sobre diferentes escenarios, los resultados podrían ser aplicables a empresas o servidores que alberguen información cuyo tiempo de carga pueda ser crucial (como por ejemplo páginas para realizar operaciones en mercados secundarios versátiles de bolsa, páginas para comprar tickets de conciertos populares en los que normalmente te dejan en espera según el tiempo de acceso a la página, comparadores de precios…).

Imaginemos el escenario de los comparadores de precios: cuando realizas una búsqueda en un comparador, éste realiza peticiones a las distintas páginas que quiera comparar y te devuelve los resultados, normalmente después de unos cuantos segundos o incluso minutos. Una vez QUIC estuviera implementado en un número relevante de servidores, escoger entre el uso de TCP y TLS o QUIC en función de cuál demuestre ser más rápido, y por tanto reducir el tiempo de las peticiones y respuestas, puede conllevar unos segundos de diferencia muy preciados por los usuarios, que pueden dar ventaja a un comparador frente a sus competidores.

Por otro lado, también podría tener efectos en cuanto a la seguridad de páginas web. Debido a que QUIC aún está en desarrollo y no tiene tantas vulnerabilidades encontradas como TCP y TLS, que llevan años en uso, habría que investigar mucho más a fondo cómo de segura realmente es la comunicación con este protocolo para atreverse a hacer una afirmación consistente. De todas maneras, intentaremos dar una visión basándonos en la teoría y en las definiciones de los protocolos de seguridad analizados.

Como comentamos en el apartado 6.3 y 6.7 de este documento, sobre la teoría QUIC debería ser más seguro debido a que encripta toda la comunicación y se autentica en servidor y cliente, incluyendo las cabeceras. Si ese fuera el caso, podría influir sobre empresas para las que la seguridad sea crucial utilizando este protocolo en vez de TCP y TLS.

4.5. Organización del documento

Este documento se organizará en distintas partes:

Primero, procederemos a explicar en detalle ambos protocolos, QUIC y HTTP/2 con sus características para ser capaz de entender mejor cómo trabajan y tenerlo en cuenta a la hora de realizar las pruebas.

Después, explicaremos los trabajos previos relacionados con comparativas entre QUIC y SPDY (el antecesor a HTTP/2), analizando sus resultados para hacernos una idea de los resultados esperados.

Seguidamente, explicaremos cómo hemos desarrollado la comparativa, explicando los diferentes escenarios de red utilizados, la automatización del proceso mediante dos scripts en el lenguaje de programación Python y cómo hemos medido el tiempo de carga y la confianza de los datos en el caso de ambos protocolos.

Por último, analizaremos los resultados y expondremos las conclusiones derivadas de los mismos.

Al final del documento se podrá encontrar la bibliografía con todas las fuentes de información, así como el anexo con la lista de páginas web que hemos utilizado para realizar la comparativa, el diagrama de Gantt del proyecto y los dos scripts utilizados para la automatización de las pruebas.

5. El protocolo HTTP/2

HTTP/2 (*Hypertext Transfer Protocol versión 2*) es un protocolo que funciona sobre TCP, diseñado para actualizar a HTTP/1.1, siendo aún compatible con este último al no modificar la semántica [17]. Los inicios de este protocolo se remontan a SPDY [18], un protocolo en desarrollo propuesto por Google cuyo objetivo era reducir la latencia de HTTP/1.1, aunque más adelante SPDY sería abandonado a favor de HTTP/2 [19], que se convertiría en un estándar en mayo de 2015, con la RFC 7540 [4].

HTTP/1.0 sólo permitía una petición por conexión TCP [20]. HTTP/1.1 intentó mejorar el rendimiento añadiendo una arquitectura de *pipeline*, pero aún sufría de bloqueo *"Head-Of-Line"* [21]. Ambos protocolos necesitaban realizar muchas peticiones a un servidor para descargar todos sus contenidos, lo que derivaba en muchas conexiones TCP simultáneas.

Además, las cabeceras de HTTP antes de la llegada de HTTP/2 eran repetitivas y detalladas, lo que causaba un uso de red innecesario y que la ventana de congestión de TCP se llenara rápidamente, resultando en un tiempo de latencia excesivo [17].

HTTP/2 optimiza la semántica de HTTP/1.1, permitiendo intercalar mensajes de solicitud y respuesta en una misma conexión TCP y codificando las cabeceras HTTP de manera más eficiente. Además, permite priorizar las solicitudes, haciendo así que sea posible recibir respuestas rápidamente para peticiones con mayor prioridad [4].

5.1 *Frames* de HTTP/2.

La unidad básica en el protocolo HTTP/2 es un *frame.* Cada uno tiene un propósito diferente: HEADERS envía las cabeceras de las peticiones y respuestas de HTTP y DATA envía los datos de las peticiones y respuestas. Un *frame* se compone de 24 bits de longitud, 8 bits para el tipo de *frame*, 8 bits para los *flags*, 1 bit reservado, 31 bits para el identificador del flujo y el resto de bits para la información [4]. En la figura 1 se muestra el formato de un *frame*:

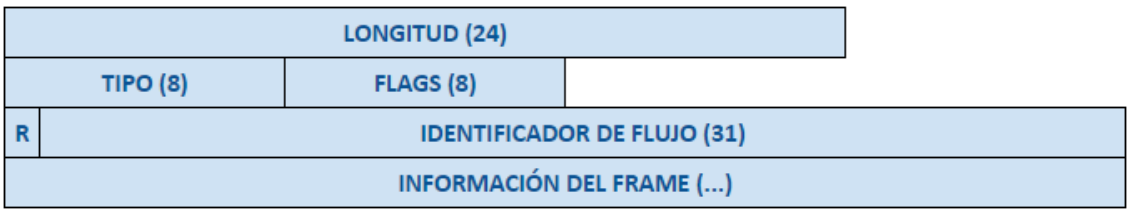


Figura 1 - Formato de un frame de HTTP/2.

5.2 Establecimiento de conexión

5.2.1. TCP

TCP tiene un establecimiento de conexión conocido como "*Three-way handshake*" (negociación a tres vías) [22].

Este mecanismo consiste en que el cliente TCP envía un *frame* SYN (*Synchronize*) al servidor para informarle de que quiere iniciar una conexión. Una vez recibido, el servidor envía al cliente un *frame* SYN-ACK (*Synchronize-Acknowledgement*), al que el cliente responderá con un *frame* ACK (*Acknowledgement*) para terminar de establecer la conexión.

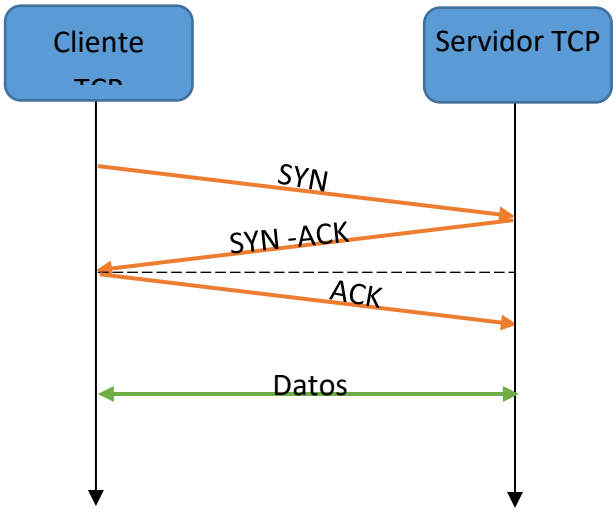


Figura 2 - Establecimiento de conexión TCP

5.2.2. TLS

En cuanto a TLS, para establecer una conexión, se sigue el siguiente proceso [23]:

1. El cliente debe iniciar la comunicación mandando un mensaje *Client Hello* que contiene la versión de TLS del cliente, el conjunto de cifrado, el método de compresión, el identificador de la sesión y un número generado aleatoriamente que servirá a posteriori para crear la clave con la que se encriptarán los datos de la comunicación.
2. Después, el servidor responderá con un mensaje *Server Hello*, aceptando las opciones del cliente.
3. Para generar las claves de encriptación, el servidor manda un mensaje llamado *Server Certificate*, adjuntando el certificado del servidor, que puede ir acompañado de un mensaje *Server Key Exchange* en caso de ser necesario (si el servidor no tiene certificado o si es sólo para firmar). Si el servidor está autenticado, es posible que

requiera un certificado del cliente. Cuando el certificado haya terminado de transmitirse, el servidor enviará un paquete indicándolo, llamado *Server Hello Done*.

4. Si el servidor ha requerido un certificado por parte del cliente, este mandará un mensaje *Client Certificate*. Después, el cliente manda un mensaje *Client Key Exchange,* afirmando que el certificado es seguro después de haberlo confirmado con una Autoridad Certificadora (CA) y creando una clave pre-maestra que encriptará usando la clave pública del servidor (extraída del certificado). El servidor descifrará esta clave usando su clave privada. Esta clave maestra será la que se utilice para toda la comunicación posterior.
5. Ahora que tanto el servidor como el cliente pueden utilizar una conexión segura y encriptada, utilizan el protocolo *Change Cipher Spec* para cambiar el cifrado al que ambos han acordado. Para esto, el cliente manda un mensaje *Change Cipher Spec* y enviará un mensaje *Finished* bajo el nuevo cifrado acordado.
6. Por último, el cliente le responde con otro *Change Cipher Spec* aceptando el cifrado y su mensaje *Finished*.

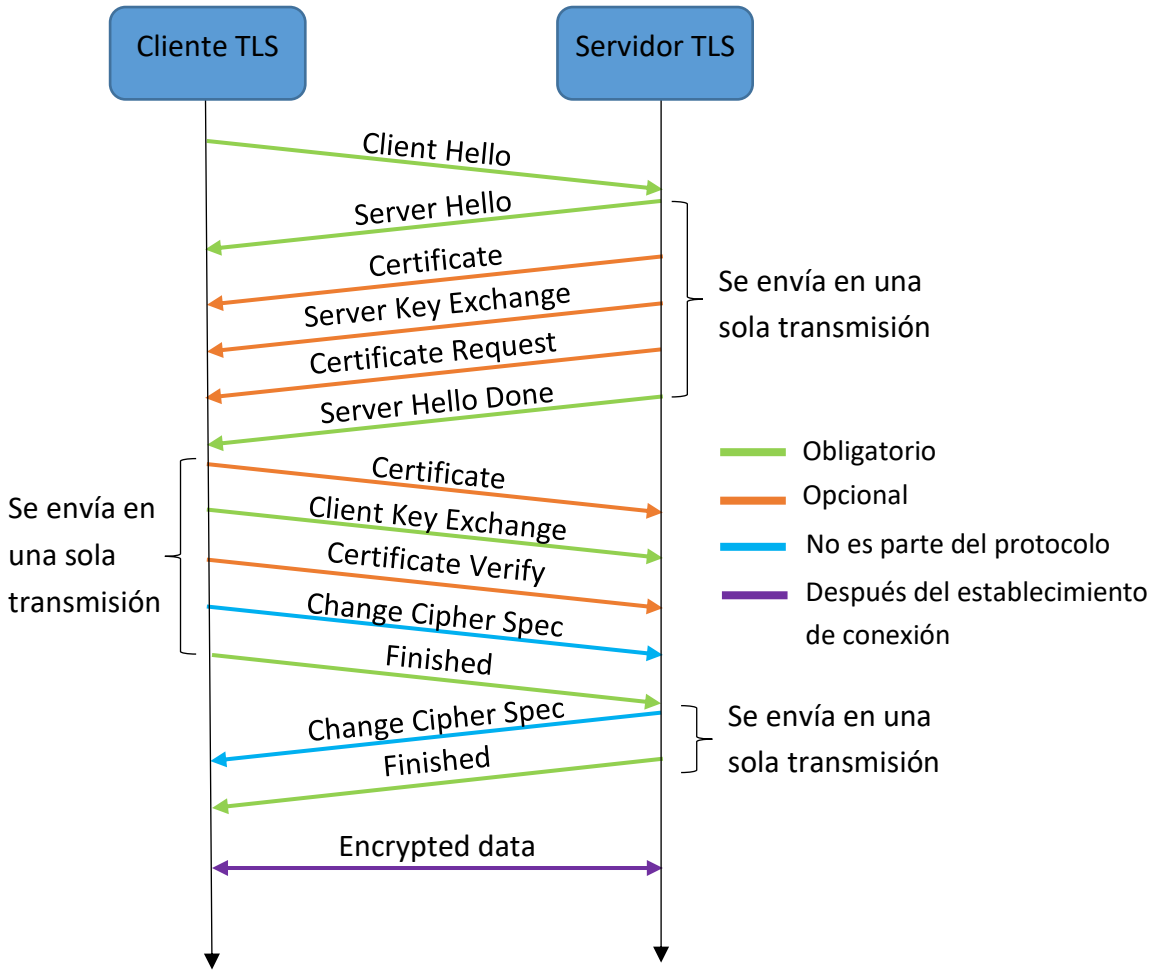


Figura 3 - Establecimiento de conexión en TLS.

En caso de que se quiera recuperar o duplicar una sesión existente, es posible reducir la negociación de la siguiente manera [24]:

1. El cliente manda un mensaje *Client Hello* con el identificador de la sesión que se desea recuperar.
2. El servidor comprueba si la sesión existe en su caché y si la encuentra, enviará un *Server Hello* con el mismo identificador de sesión.
3. Después, se enviarán los respectivos mensajes *Change Cipher Suite* y *Finish* y la conexión estará reestablecida.

5.3 Una única conexión TCP.

Con HTTP/1.0, se establecía una conexión TCP por cada petición que se hiciera al servidor. Dado que muchas páginas utilizan orígenes distintos, una única página podría crear más de treinta conexiones TCP distintas, pudiendo llegar a monopolizar los recursos de red.

Esta sobrecarga de los recursos de red se conoce como *"Head-of-line blocking"* (HOL). Este problema surge cuando muchos paquetes están bloqueados debido a que un paquete anterior está bloqueando la conexión.

Debido a que TCP requiere entrega ordenada, si un paquete se perdía o tardaba mucho en enviarse, bloqueaba al resto de paquetes de una misma petición o respuesta haciendo que el tiempo de latencia incrementara bastante. En HTTP/1.x, esto ocurría debido a que sólo una petición podía ser atendida de manera eficiente en un periodo de tiempo determinado, lo que obligaba a esperar a la respuesta del servidor para poder mandar la siguiente petición [25].

HTTP/1.1 intentó solucionar esto mediante una arquitectura de *pipeline,* aunque una respuesta larga o lenta podía seguir bloqueando la comunicación, además de que es una arquitectura difícil de desplegar ya que muchos servidores no la procesan correctamente, debido a que, pese a que se requiere que soporten *pipeline*, esto no significa que utilicen este mecanismo para enviar las respuestas, sino que no fallarán en caso de que un cliente escoja mandar las peticiones con esta arquitectura [25].

Con esta solución, para los navegadores era posible establecer una única conexión, aunque por lo general se establecían entre cuatro y ocho conexiones por cada origen de datos para mayor eficiencia.

HTTP/2 corrige este problema gracias a la multiplexación, que permite responder a las peticiones en un orden distinto al de llegada, incluso pudiendo entrelazar flujos de respuestas diferentes debido a que cada frame de HTTP/2 lleva un campo de

identificador de flujo. Si un flujo se bloquea, no bloquea el progreso del resto de flujos, lo que permite reducir el número de conexiones y mejora el tiempo de carga [4].

En la siguiente figura se ilustra la multiplexación en HTTP/2.

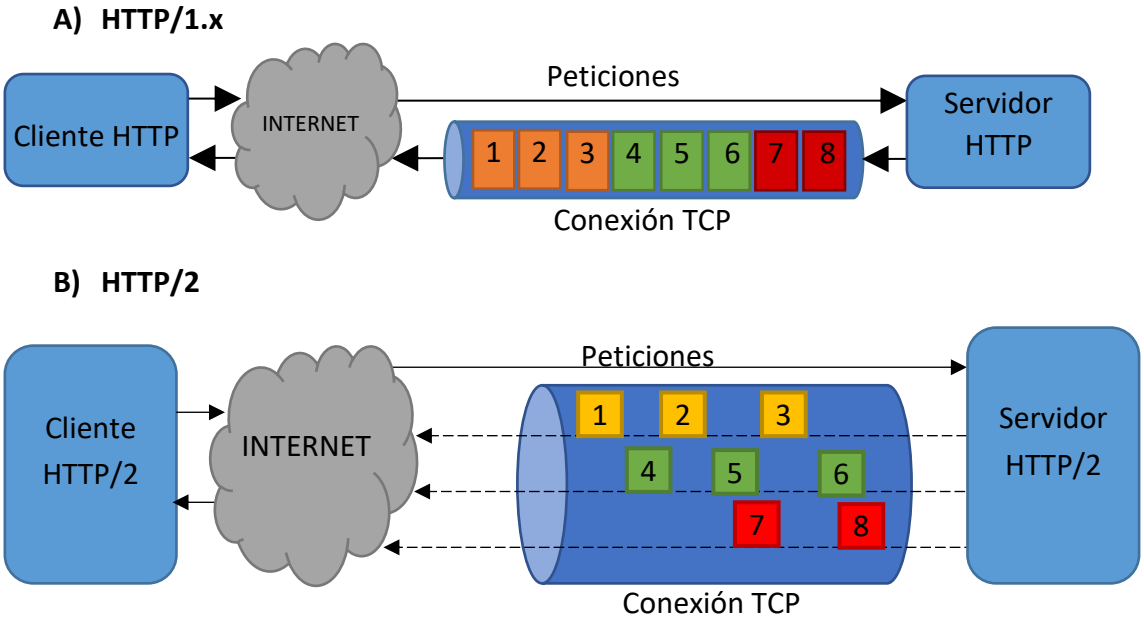


Figura 4 - Multiplexación en HTTP/2

5.4 Un protocolo binario

Los protocolos binarios son más eficientes para analizar sintácticamente, son más compactos y menos propensos a errores, comparados con los protocolos textuales como HTTP/1.x [4].

Dado que HTTP/2 es un protocolo binario, es más sencillo encontrar el comienzo y final de cada *frame*.

5.5 Priorización y control de flujos

Una petición o respuesta HTTP puede dividirse en varias partes en el transporte del servidor al cliente o viceversa. Dado que hay elementos de una página que son más importantes que otros, la priorización de los flujos es importante en HTTP/2.

El mecanismo de priorización se basa en términos de dependencia y pesos de flujos. Un flujo puede tener dependencia de otro flujo (lo que le indica al servidor que debería asignar recursos al flujo identificado en vez de al dependiente, a menos que el flujo identificado ya esté cerrado o no pueda progresar) [26].

Los flujos dependientes también tienen asignados un peso, entre 1 y 256, y los recursos serán asignados de manera proporcional a su peso.

Respecto al control de flujos, HTTP/2 debe asegurarse de que los diversos flujos que se envían sobre una conexión TCP no interfieran con los demás de manera destructiva. El control de flujo se basa en el *frame* `WINDOW_UPDATE` (el receptor anuncia por adelantado cuántos octetos pueden recibir en un flujo o en una conexión) y es específico a una conexión. Además, el emisor debe respetar siempre el valor del tamaño de la ventana, el control de flujo nunca puede deshabilitarse y el tipo de *frame* determina si el control de flujo se aplica a ese *frame* determinado. El valor inicial de la ventana para el control de flujo es de 65.535 octetos [4].

5.6 Compresión de cabeceras

En las versiones anteriores de HTTP, las cabeceras se transmitían en texto claro, aumentaban su tamaño considerablemente gracias al uso de cookies, a los *user-agent* de navegadores… Además, las cabeceras de peticiones a un mismo servidor y respuestas a un mismo cliente no suelen variar mucho, por lo que se enviaba mucha información redundante.

HTTP/2 introduce una forma de compresión de cabeceras conocida como HPACK, que permite reducir las cabeceras sin ser vulnerable a los ataques relacionados con la compresión.

Una lista de cabeceras es un conjunto de uno o más campos de una cabecera. Al transmitirla a través de una conexión, se serializa utilizando HPACK y se divide en varias secuencias de octetos y se transmiten con la información de `HEADERS`, `PUSH_PROMISE` o `CONTINUE` [27].

Debido a la compresión de cabeceras es posible codificar cabeceras de gran tamaño y cabeceras comunes como una variable de tipo entero (en vez de reenviar toda la cabecera de nuevo).

5.7 Servicio "*server push*"

El servicio "*server push*" consiste en estimaciones para que el servidor envíe información al cliente sin haber recibido una petición explícita previamente por parte del cliente para conseguir disponibilidad inmediata de la información [28].

Esto es especialmente útil debido a que una aplicación web suele consistir en numerosos recursos diferentes, que el cliente descubre a medida que va examinando la respuesta principal del servidor. En las versiones anteriores el cliente debía hacer una petición por cada uno de los recursos necesarios, mientras que con HTTP/2, el servidor es capaz de mandárselos después de la respuesta a la petición original para ahorrar tiempo de latencia.

Además, los recursos enviados con el *"server push"*, llamados *"promises"* tienen beneficios adicionales como poder ser cacheados, reutilizados en diferentes páginas, priorizados por el servidor o declinados por el cliente [4].

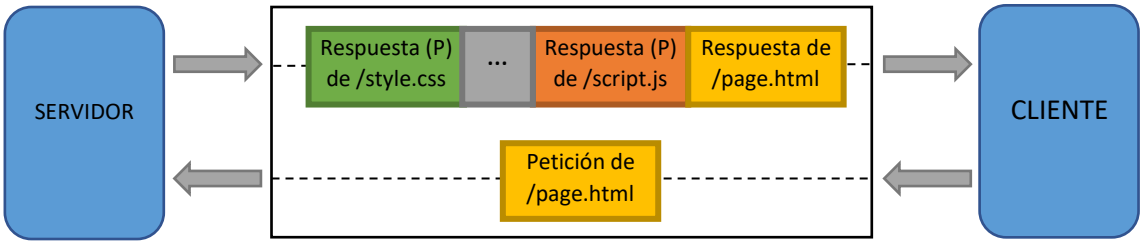


Figura 5 - Servicio server push

6. El protocolo experimental QUIC.

El protocolo QUIC (*Quick UDP Internet Connections*) es un protocolo experimental de la capa de transporte diseñado por Google que, a diferencia de otros protocolos, soporta un conjunto de conexiones multiplexadas sobre UDP. La intención de este protocolo es proveer la seguridad equivalente a TLS/SSL mientras mejora el rendimiento del protocolo HTTP sobre TCP y TLS, reduciendo el tiempo de conexión y transporte [3].

En resumen, QUIC provee multiplexación y control de flujo equivalente a HTTP/2, una seguridad equivalente a TLS y una semántica de conexión, confiabilidad y control de congestión equivalente a TCP.

Actualmente, Google ha desplegado el protocolo en sus servidores y tiene una implementación de cliente en los navegadores web de Chrome [29].

6.1. Paquetes en QUIC

La unidad básica del protocolo QUIC es un paquete. Existen paquetes regulares o especiales: los paquetes regulares contienen *frames*, mientras que los paquetes especiales pueden ser de negociación de versión o de reseteo público.

Todos los paquetes de QUIC deben ser de un tamaño que encaje con el MTU (*Maximum Transmission Unit*) para evitar la fragmentación IP. Actualmente, QUIC utiliza una MTU de 1350 bytes para IPv6 y de 1370 bytes para IPv4.

Los paquetes en QUIC empiezan con una cabecera que ocupa entre 2 y 19 bytes. Dicha cabecera consta de 8 bits para *flags* públicos, entre 0 y 64 bits para el identificador de conexión, 32 bits opcionales para la versión de QUIC, y 8, 16, 32 o 48 bits para el número del paquete [5].

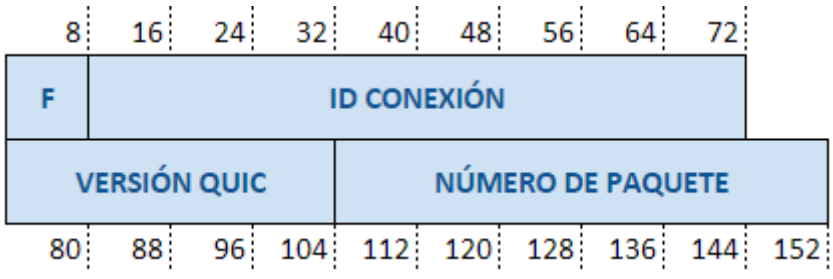


Figura 6 - Formato de un paquete en QUIC

6.2. Multiplexación

Como hemos comentado al hablar sobre HTTP, el hecho de que sea un protocolo que utilice TCP como protocolo de transporte lo hace vulnerable al bloqueo "*Head of Line*".

UDP es un protocolo de transporte que no exige entrega ordenada, por lo que en caso de que un paquete se pierda, el resto de paquetes pueden ser entregados sin verse afectados.

Como QUIC utiliza este protocolo para la capa de transporte, la multiplexación en HTTP/2 sobre QUIC resulta más efectiva que en HTTP/2 sobre TCP y TLS, ya que en caso de que un paquete que contenga información sobre un flujo específico se pierda, no afectará al resto de flujos. De esta manera, la transmisión de paquetes continúa en los flujos que no hayan sufrido pérdidas de paquetes, pudiéndose juntar de nuevo en el cliente y seguir haciendo progresos en la descarga de la página [5].

Figura 7 - Multiplexación en QUIC

Como QUIC relega la compresión de cabeceras a HTTP/2 mediante la compresión HPACK en el flujo de cabeceras, es posible que exista bloqueo "*Head of Line*" en este tipo de *frames* únicamente.

6.3. Establecimiento de conexión

Como hemos visto en el apartado 5.2, en HTTP/2, para establecer una conexión, se necesitarán mínimo un RTT para el establecimiento de conexión TCP y uno o dos más para la negociación TLS antes de enviar cualquier dato útil. Aunque la RFC permite conexiones en claro (sin utilizar TLS), la mayoría de navegadores no soportan esta opción lo que incrementa notablemente el tiempo total de la petición.

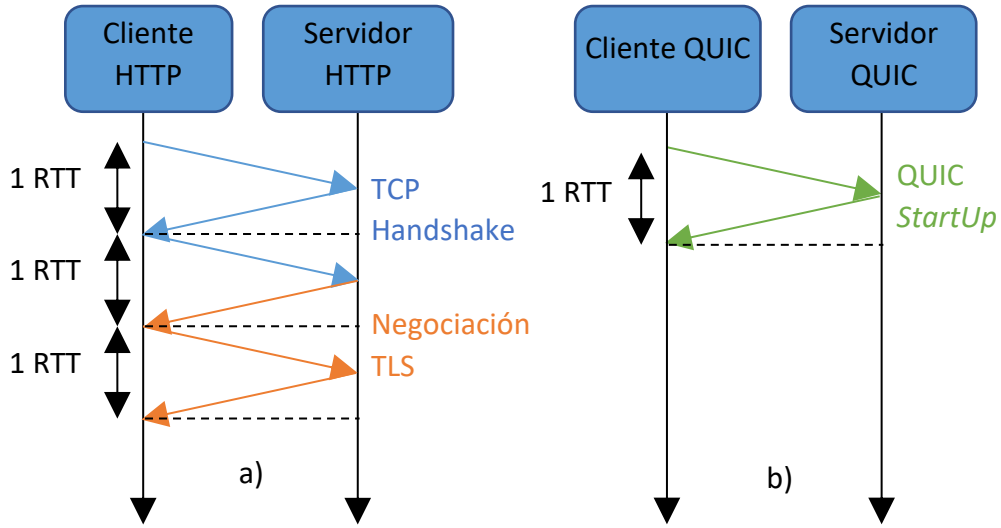


Figura 8 - Latencia de inicio

QUIC pretende reemplazar TCP y TLS con UDP y la propia seguridad de QUIC para así conseguir 0 RTTs.

La primera vez que un cliente QUIC se conecta al servidor, el cliente enviará un mensaje *Client Hello* (CHLO) vacío, y el servidor enviará una respuesta *Rejection* (REJ) con la información necesaria para el cliente, incluyendo el *token* fuente de dirección y los certificados del servidor. Como en TLS, esto sólo será necesario la primera vez, ya que las conexiones consecutivas funcionarán con credenciales cacheadas de la conexión anterior [5].

A partir de esa primera conexión, podremos conseguir un establecimiento de conexión con 0-RTTs: una vez se haya verificado el servidor (tras recibir el REJ), el cliente mandará un *Client Hello* (en caso de querer conseguir 0-RTTs, este mensaje puede contener peticiones) y el servidor de devolverá un *Server Hello* (SHLO) que podrá contener las respuestas a las peticiones del *Client Hello*.

Este mecanismo de establecimiento de conexión es temporal, ya que será reemplazado por la versión de TLS 1.3 que implementará de por sí el modo de 0-RTTs [30].

No.	Time	Source	Destinati	Protocol	Len	Info
613	5.1184723…	192.168.0.198	216.…	QUIC	1…	Client Hello, PKN: 1, CID: 4327597967259372…
620	5.1661733…	216.58.205.100	192.…	QUIC	1…	Rejection, PKN: 1, CID: 4327597967259372521
621	5.1684194…	216.58.205.100	192.…	QUIC	1…	Payload (Encrypted), PKN: 2, CID: 432759796…
622	5.1714481…	192.168.0.198	216.…	QUIC	82	Payload (Encrypted), PKN: 2, CID: 432759796…

Figura 9 - Establecimiento de conexión por primera vez en QUIC

En cuanto a la seguridad, QUIC utiliza su propio algoritmo de encriptación, llamado QUIC-Crypto, que desencripta paquetes independientemente del resto, evitando la dependencia de decodificación serializada que podría dificultar la habilidad de QUIC de proporcionar la entrega desordenada de paquetes [31].

6.4. Corrección de errores hacia adelante

Una característica importante de QUIC es la corrección de errores hacia delante, o *Forward Error Correction* en inglés. Esta característica beneficia a QUIC reduciendo los posibles bloqueos *"Head of Line"* en un flujo QUIC, especialmente en el caso de RTTs altos (donde una retransmisión podría afectar gravemente al tiempo total de la transmisión), recuperando un paquete perdido sin necesidad de retransmitir.

La corrección de errores hacia adelante se basa en un algoritmo básico XOR. Dentro de un mismo flujo de paquetes, se envía un paquete FEC que contendrá la paridad de un grupo de paquetes determinado (el número de paquetes del grupo será decidido por el emisor para optimizar la transmisión). Si uno de los paquetes del grupo se pierde, su contenido podrá ser recuperado mediante un análisis del paquete FEC junto con el resto de los paquetes del grupo [32].

6.5. Control flexible de congestión

QUIC ha sido diseñado para soportar dos algoritmos de congestión de control, decidiendo en la negociación al inicio de la conexión qué mecanismo usar.

Por defecto, QUIC utiliza TCP Cubic, aunque ofrece un mecanismo alternativo, más completo que el ofrecido por TCP, por lo que proporciona más información para el control de congestión.

Por ejemplo, en QUIC cada paquete, ya sea original o una retransmisión llevan un número de secuencia nuevo. De esta manera, el emisor podría diferenciar qué ACKs indican que debe transmitir un nuevo paquete o cuales indican que debe retransmitir un paquete anterior. Además, un ACK en QUIC también muestra la diferencia de tiempo entre la recepción del paquete y el ACK por parte del receptor. Gracias a estas dos características, es posible calcular de manera bastante precisa el RTT.

En QUIC, cuando un temporizador expira, se realizan las siguientes acciones [5]:

- Si está en la fase de establecimiento de conexión, se retransmite cualquier paquete de establecimiento de conexión.
- Si expira el temporizador de pérdidas: se descartan todos los paquetes que no hayan recibido un ACK, se reporta la pérdida de esos paquetes al controlador de congestión y se retransmiten tantos como el controlador de congestión admita.
- Si expira el temporizador TLP: Se retransmite el paquete más pequeño sin ACK, no se marca ningún paquete como perdido hasta que llegue un ACK nuevo y se reinicia el temporizador.

- Si expira el temporizador RTO: Se retransmiten los dos paquetes más pequeños sin ACK, no se colapsa la ventana de congestión hasta que llegue un ACK y se reinicia el temporizador para el siguiente RTO.

6.6. Control de flujo de conexión

QUIC sigue de cerca el control de flujo de HTTP/2: Un receptor de paquetes de QUIC anuncia el offset de bytes en cada flujo indicando la cantidad de datos que está dispuesto a aceptar del emisor. Según se va recibiendo la información de un flujo particular, el receptor envía *frames* de tipo `WINDOW_UPDATE` actualizando el límite de datos y permitiendo al emisor que incremente el número de bytes que envía por paquete [33].

Además, QUIC implementa un control de flujo a nivel de conexión que limita el búfer que un receptor QUIC está dispuesto a asignar a una conexión. El control de flujo funciona de manera similar al de un flujo particular, con la diferencia de que el tamaño del buffer es común a todos los flujos dentro de una misma conexión.

6.7. Prevención contra ataques de inyección y manipulación de cabeceras en QUIC

Las cabeceras de TCP por lo general se muestran en claro en conexiones por cable y no están autenticadas, lo que genera muchos ataques de inyección y manipulación de cabeceras [5].

Los paquetes de QUIC siempre están autenticados y su información por lo general está totalmente encriptada. Aquellas partes de las cabeceras que no estén encriptadas se autentican por el receptor, por lo que cualquier ataque de terceros de inyección o manipulación de cabeceras se vería frustrado. Los únicos paquetes que no sufren estas medidas de seguridad son los de tipo `PUBLIC_RESET` que sirve para resetear una conexión.

6.8. Migración de la conexión

Un problema existente en TCP es que, dado que sus conexiones estaban identificadas por el puerto e IP de destino y origen, las conexiones no sobreviven a cambios en la dirección IP (si por ejemplo el terminal cambia de una conexión inalámbrica Wi-Fi a una por satélite) o a cambios de puerto [5].

Como las conexiones en QUIC están identificadas por el identificador de conexión, puede sobrevivir a cambios en la dirección IP o en el puerto debido a que ese identificador no cambia con migraciones de conexión.

Además, QUIC también provee verificación cifrada automática de un cliente en migración, ya que se sigue usando la misma clave de sesión para el cifrado y descifrado de paquetes.

6.9. HTTP/2 sobre QUIC

En este proyecto estudiamos el impacto en el rendimiento de HTTP/2 sobre QUIC. Debido a que QUIC tiene que soportar a HTTP/2 como protocolo superior, adapta sus características a este protocolo. En los siguientes sub-apartados estudiaremos cómo se ejecuta HTTP/2 sobre este protocolo [5].

6.9.1. Administración de los flujos

En el caso en el que HTTP/2 se ejecuta sobre QUIC, relega a este toda la administración de flujos, es decir, se ejecuta el control de flujo de QUIC y no el definido por HTTP/2 para TCP y TLS.

Para ello, los identificadores de flujo de HTTP/2 se reemplazan por los de QUIC, y no es necesario que HTTP/2 haga ningún cambio en su estructura cuando utiliza QUIC como protocolo de transporte: los datos que se envían en los paquetes de QUIC son directamente las cabeceras o el cuerpo de HTTP/2.

Una petición o respuesta se considera completada cuando el flujo QUIC se cierra en esa dirección.

6.9.2. Compresión de cabeceras HTTP/2

Como hemos adelantado antes, QUIC utiliza la compresión de cabeceras definida en HTTP/2 (HPACK), lo que deriva en problemas de bloqueo *"Head of line"* debido a que los bloques que contengan cabeceras deben ser descomprimidas en el mismo orden en el que se comprimieron [5].

Como el uso de flujos permite que estos sean procesados en cualquier orden en la máquina del receptor, QUIC fuerza un orden estricto para las cabeceras utilizando un solo flujo para enviar todos los bloques con cabeceras (el flujo con identificador número 3). De esta manera, un receptor HTTP/2 que use QUIC como protocolo de transporte procesará los datos de un flujo determinado sólo cuando haya recibido la correspondiente cabecera en el flujo de cabeceras.

Se espera que QUIC implemente un mecanismo para compresión y descompresión de cabeceras para que no sea necesaria una descompresión ordenada de las cabeceras y así poder evitar o reducir el bloqueo *"Head of line"*.

6.9.3. Negociación de QUIC en HTTP

Para que HTTP/2 utilice QUIC como protocolo de transporte, es necesario que el servidor lo especifique como un protocolo alternativo en el puerto en el que se vaya a usar [5]. Si queremos usar QUIC sobre una conexión segura, utilizamos:

```
Alternate-Protocol: 443:quic
```

Cuando un cliente recibe esta cabecera, intentará usar QUIC para las siguientes conexiones en ese dominio.

Como algunos firewalls pueden bloquear conexiones de QUIC o UDP, es necesario configurar el cliente para que, en caso de no estar disponible, relegue a TCP de nuevo.

Es posible también utilizar la cabecera `Alternate-protocol-required` para notificar al cliente que no se permite el uso de QUIC en ese dominio. Cuando el cliente reciba esta cabecera, no volverá a comprobar si QUIC está disponible y no volverá a usar QUIC en ese dominio.

7. Trabajo previo

Debido a que QUIC lleva en desarrollo desde 2012, existen varios artículos e investigaciones que miden su rendimiento comparándolo con SPDY, el protocolo predecesor de HTTP/2.

En particular, en 2014, Somak R. Das, del Massachusetts Institute of Technology (MIT), realizó un Trabajo de Fin de Master llamado "*Evaluation of QUIC on Web Page Performance*" [7], en el que comparaba este nuevo protocolo con HTTP/1.1 y SPDY, llegando a la conclusión de que QUIC tenía un rendimiento de 1 o 2 veces menor que SPDY, aunque sin embargo mejoraba el rendimiento de HTTP/1.1 en enlaces con alto RTT, así como en enlaces con bajo ancho de banda.

En 2015, Gaetano Carlucci, Luca De Cicco y Saverio Mascolo, del Politecnico di Bari & Quavlive, en Italia, redactaron un artículo llamado "*HTTP over UDP: an Experimental Investigation of QUIC*" [34], en el que llegaban a la conclusión de que con QUIC se reducía el tiempo total de carga de una página web en comparación con el uso de HTTP/1.1 en un canal en el que no existen pérdidas. Además, mejoraba a SPDY en el caso de un medio con pérdidas. También llegaron a la conclusión de que activar el módulo FEC afectaba negativamente al rendimiento de QUIC.

Por último, en 2016 Péter Megyesi, Zsolt Krämer y Sándor Molnár, del departamento de Telecomunicaciones e Informática de Budapest University of Technology and Economics, en Hungría, hicieron un simposio llamado "*How quick is QUIC?*" [35], llegando a la conclusión de que los tres protocolos (HTTP/1.1, SPDY y QUIC) actuaban parecido bajo unas buenas condiciones de red (alto ancho de banda, bajo RTT y pocas pérdidas de paquetes) en el caso de páginas web de tamaños pequeños y medianos. En el caso de páginas más pesadas, llegando a la conclusión de que QUIC no tiene un buen rendimiento en el caso de enlaces de alta velocidad (debido a que no consigue aprovechar toda la capacidad del enlace). Además, concluían que en redes con altas pérdidas el módulo FEC de QUIC funcionaba bastante bien.

Como hemos visto, estos tres estudios ponen a prueba QUIC (con los cambios realizados año tras año), aunque lo comparan con SPDY y HTTP/1.1, no con HTTP/2. Con este estudio pretendemos actualizar estos resultados con los dos protocolos más actualizados.

8. Diseño de la comparativa

El objetivo de este estudio es comparar los tiempos de carga de una página web utilizando HTTP/2 sobre el protocolo QUIC y compararlo con el uso de HTTP/2 sobre TCP y TLS.

8.1. Explicación de las pruebas

Para comparar ambos protocolos, realizamos diez descargas sobre cincuenta y tres páginas web en dieciséis escenarios de red distintos utilizando ambos protocolos y medimos el tiempo de carga de ambos.

Debido a que los únicos servidores web que entienden el protocolo QUIC son aquellos que son propiedad de Google e incluyen en sus respuestas la cabecera `Alternate-Protocol: 443:quic`, hemos realizado el experimento utilizando una lista de los diferentes servicios de Google, dando lugar a esas cincuenta y tres páginas en las que hemos probado el experimento. Dicha lista se puede encontrar en el Anexo I de este documento.

Hemos decidido probar tantas páginas como nos fuera posible dadas las limitaciones debido a que las páginas web pueden variar mucho unas de otras en cuanto a peso de la página o cantidad de contenido, modificando así el tiempo de carga de unas a otras, por lo que intentaremos que en esas cincuenta y tres páginas haya todo tipo de combinaciones entre: páginas ligeras o pesadas y páginas con mucho y poco contenido.

El tiempo de carga de una página web puede variar dependiendo de las características de la red que se usen (ancho de banda, pérdidas de paquetes, latencia…), por lo que realizaremos las pruebas emulando distintos perfiles de red modificando la latencia, el ancho de banda y el porcentaje de pérdidas de paquetes para observar cómo se comportan ambos protocolos en cada caso.

8.2. Escenarios utilizados

En este estudio, hemos probado con 16 configuraciones diferentes:

- Velocidad de descarga y subida de 0,6 Mbps:
  - Latencia de 30 milisegundos:
    - Porcentaje de pérdidas: 0%.
    - Porcentaje de pérdidas: 0.1%.
    - Porcentaje de pérdidas: 1%.
    - Porcentaje de pérdidas: 10%.
  - Latencia de 270 milisegundos.
    - Porcentaje de pérdidas: 0%.

        - Porcentaje de pérdidas: 0.1%.
        - Porcentaje de pérdidas: 1%.
        - Porcentaje de pérdidas: 10%.
- Velocidad de descarga y subida de 12 Mbps:
    - Latencia de 30 milisegundos:
        - Porcentaje de pérdidas: 0%.
        - Porcentaje de pérdidas: 0.1%.
        - Porcentaje de pérdidas: 1%.
        - Porcentaje de pérdidas: 10%.
    - Latencia de 270 milisegundos.
        - Porcentaje de pérdidas: 0%.
        - Porcentaje de pérdidas: 0.1%.
        - Porcentaje de pérdidas: 1%.
        - Porcentaje de pérdidas: 10%.

8.3 Automatización de las pruebas

Debido a que la realización de diez cargas sobre cincuenta y tres páginas en cada uno de estos dieciséis escenarios suma un total de 16.960 pruebas, nos hemos basado en un script en Python [6] desarrollado por Somak R. Das para su estudio *"Evaluation of QUIC on Web Page Performance"* [7] (ver apartado 7 de este documento) y lo hemos modificado para que se adapte a nuestro estudio.

Para la emulación de red hemos utilizado el programa Mahimahi [8], que te permite simular una red con un porcentaje de pérdidas a la elección del usuario (tanto para enlaces de subida como de descarga), con la latencia deseada y con una velocidad de subida y descarga definida en un fichero de texto.

Dicho fichero consta de una lista de números enteros de longitud aleatoria que indican el segundo en el que debemos permitir a Mahimahi lanzar un paquete a la red. Dicho paquete será del tamaño del MTU teniendo en cuenta todas las cabeceras de protocolos previos, es decir, 1500 bytes o 12000 bits. De esta manera, para emular una red de 12 Mbps, la lista de números tenía un intervalo de 1 entre un número y otro, permitiendo que 12000 bits se mandasen cada segundo. Para emular la red de 0,6 Mbps, la lista tenía un intervalo de 20 entre un número y otro, permitiendo que 12000 bits se manden cada 20 segundos (600 bits por segundo de media).

Para monitorizar la carga de las páginas, hemos utilizado Selenium 2.39.0 [36] junto con Chrome WebDriver 2.32 [37]. Para cada uno de los protocolos hemos especificado los *flags* necesarios para que Chrome utilice QUIC o HTTP/2 como opciones y hemos ejecutado Chrome WebDriver con esas opciones. Para que no muestre el navegador en pantalla hemos utilizado Python Virtual Display [38] con la opción `visible=0`.

Para medir el tiempo de carga hemos calculado la diferencia entre los eventos `navigationStart` (el principio de una carga tal y como lo percibe el usuario) y `loadEventEnd`, que mide el tiempo del evento `Load` en una página.

Para realizar las 16.960 pruebas, hemos utilizado dos scripts: `load_page.py` y `pruebas.py`.

8.3.1. Script load_page.py (Anexo III)

Este script está basado en el script `load_page.py` [6] diseñado por Somak R. Das, aunque hemos realizado varias modificaciones para que se adapte mejor a nuestras pruebas.

Este script realiza la carga de una página web y devuelve el tiempo de carga. Recibe como parámetros la página web a cargar, la latencia, la velocidad de subida y descarga y el porcentaje de pérdidas.

El script realizará la carga de la página, medirá la diferencia de tiempo entre `navigationStart` y `loadEventEnd` para hallar el tiempo de carga y escribirá la página web y el tiempo de carga en un documento con el siguiente formato: `tiempo_carga_<velocidad>Mbps_<latencia>ms_<porcentaje_de_pérdidas>perc.txt`.

Los parámetros que recibe como argumentos sólo sirven para crear o abrir el documento donde escribirá el tiempo de carga de la página, ya que la emulación de red se realiza en el segundo script utilizado.

Para finalizar, cerramos los procesos mediante un `driver.quit()` y `display.stop()`.

8.3.2. Script load_page.py (IV)

Este script recibe por parámetro un fichero de texto con una lista de páginas web a emular separadas entre sí por un salto de línea.

Consta de tres *arrays* con las velocidades de red, las latencias y los porcentajes de pérdidas que queremos probar y cinco bucles (cada uno dentro de los anteriores): Un bucle contador para que repita cada configuración diez veces, tres bucles que recorren los tres *arrays* con las configuraciones y otro que recorra la lista de sitios web.

```
i=1
while i<11:
    for link_speed in link_speeds:
        for delay in delays:
            for loss in losses:
                print "link_speed =", link_speed, ", delay =", delay, ",

                with open(site_list) as f:
                    file_name = "replay_summary_" + str(link_speed) + "M
                    with open( file_name, 'a' ) as rec:
                        rec.write( "----------------------"+ str(i) +"---
                        rec.close()
                    for line in f:
```

Figura 10 - Bucles en el script pruebas.py

Para cada página y configuración, primero utiliza Mahimahi [8] para crear la red emulada en función de los parámetros seleccionados en esa altura del bucle, y por último llama al script `page_load.py` [6] para que realice la carga de la página deseada.

Cada vez que acaba de realizar la carga de las cincuenta y tres páginas en todos los escenarios, escribe en el fichero creado por el script `page_load.py` [6] el número de iteración que se va a realizar para facilitar el análisis de los resultados.

8.4 Configuración para HTTP/2 sobre QUIC.

Para indicarle a Chrome WebDriver 2.32 [37] que deseamos utilizar QUIC en vez de TCP y TLS como protocolo de transporte tendremos que indicárselo mediante los *flags* de opciones que permite Google Chrome.

Para eso, definiremos cuatro *flags* distintos:

1. `--incognito`: Debido a que realizaremos muchas cargas seguidas y queremos que cargue el contenido completamente cada vez y no lo almacene y reutilice después de la memoria caché, utilizaremos este *flag* para iniciar sesión como incógnito y evitar esta situación.
2. `--ignore-certificate-errors`: Dado que la mayoría de los navegadores, incluido Chrome, no aceptan el uso de HTTP/2 si no es sobre conexiones seguras, utilizamos este *flag* para asegurarnos que nos permite siempre el uso de este protocolo, aunque haya cualquier error de certificado.
3. `--enable-quic`: Indicamos al navegador que queremos que utilice QUIC siempre que esté disponible (en nuestro caso, siempre, ya que todos los servidores a los que nos conectaremos lo aceptan).
4. `--enable-quic-https`: Permite el uso de QUIC sobre un canal encriptado y autenticado. Con esto indicamos que queremos que utilice la seguridad proporcionada por QUIC para el canal.

Para configurar Chrome WebDriver 2.32 [37] bajo estas opciones, las añadiremos como argumento a `Options` y lanzaremos el programa indicando que esas son las opciones. En la siguiente figura podemos encontrar el código utilizado:

```
options=Options()
options.add_argument("--incognito")
options.add_argument("--ignore-certificate-errors")
options.add_argument("--enable-quic") #Only to run QUIC
options.add_argument("--enable-quic-https") #Only to run QUIC
chromedriver = "/home/lbalart/Descargas/chromedriver"
driver=webdriver.Chrome(chromedriver, chrome_options=options)
```

Figura 11 - Configuración de opciones para utilizar QUIC

Antes de lanzar el script, nos aseguramos de que utiliza QUIC correctamente:

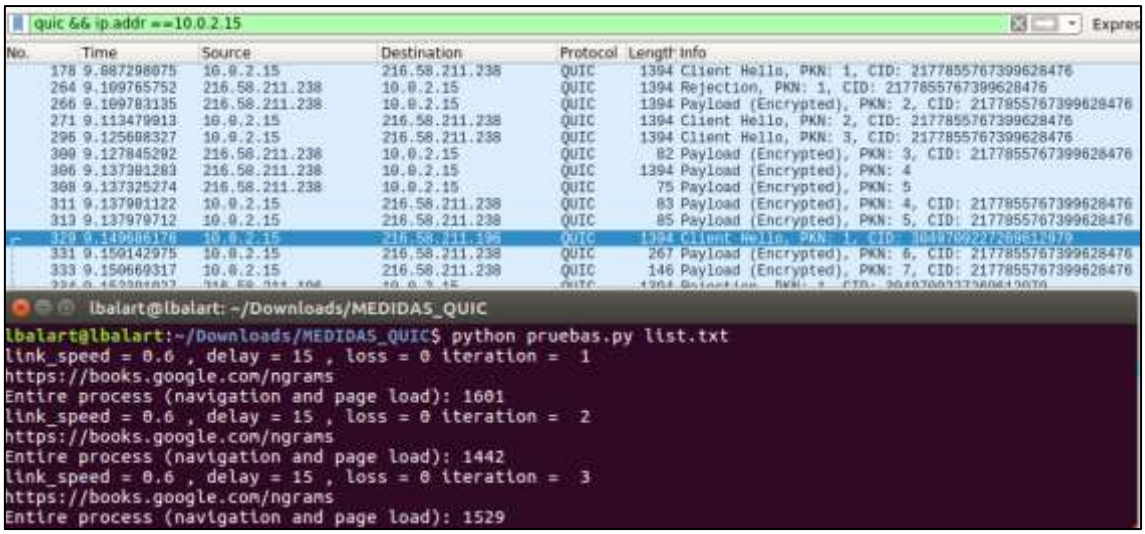


Figura 12 - Script utilizando QUIC

8.5 Configuración para HTTP/2 sobre TCP y TLS.

Google Chrome acepta tres configuraciones para el uso de QUIC: deshabilitado (no usar nunca), por defecto (usar sólo en algunos casos) o habilitado (usar siempre que el servidor lo permita).

Debido a que los servidores de Google a los que vamos a conectarnos aceptan por defecto el uso de HTTP/2 sobre TCP y TLS, simplemente tendremos que asegurarnos de que QUIC no esté habilitado en la configuración "por defecto" para que en todas las pruebas utilice TCP y TLS. Por otro lado, como Google Chrome sólo acepta el uso de HTTP/2 sobre conexiones seguras, tendremos que evitar los errores por certificados.

Para ello, volveremos a configurar las opciones, utilizando los *flags* `--ignore-certificate-errors` e `--incognito` para asegurarnos que las conexiones siempre son seguras y que no almacene y recupere datos de la memoria caché, añadiendo el *flag*

`--disable-quic` para asegurarnos que está deshabilitado, y quitaremos los *flags* relativos a QUIC (`--enable-quic` y `--enable-quic-https`).

En la siguiente figura se puede ver el código utilizado:

```
options=Options()
options.add_argument("--incognito")
options.add_argument("--ignore-certificate-errors")
options.add_argument("--disable-quic") #Only to run TCP + TLS
chromedriver = "/home/lbalart/Descargas/chromedriver"
driver=webdriver.Chrome(chromedriver, chrome_options=options)
```

Figura 13 - Configuración de opciones para utilizar TCP + TLS

Para no sobre escribir los ficheros con los datos de QUIC, creamos otra carpeta llamada `MEDIDAS_HTTP` donde crearemos de nuevo los ficheros y donde nos localizaremos en la consola para lanzar el script.

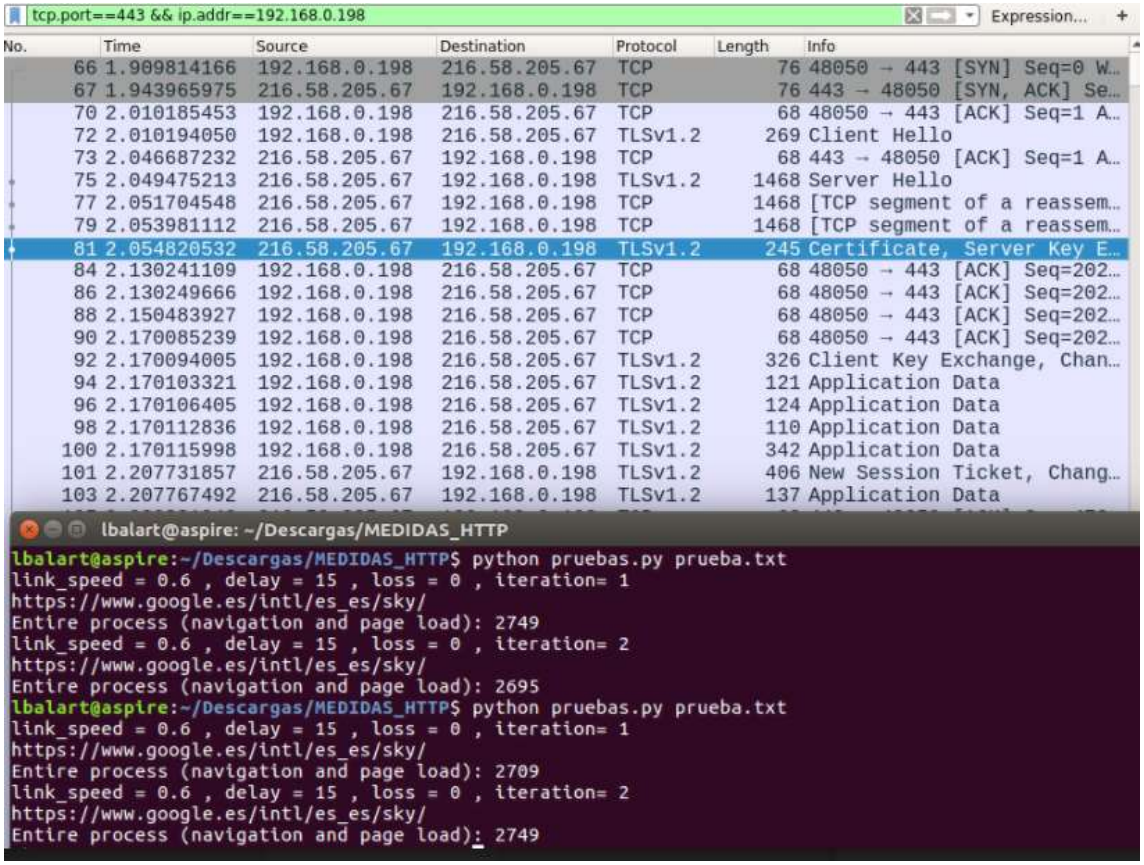


Figura 14 - Script utilizando TCP y TSL

8.6. Procesamiento de resultados.

Una vez tenemos los 32 ficheros generados (16 para los escenarios con QUIC y otros 16 para los escenarios con TCP y TLS), utilizaremos el programa Microsoft Excel [39] para analizar los datos.

Para ello, hemos creado 16 hojas de un mismo libro (una para cada escenario de red, e introducido los datos mediante la opción de “Obtener datos externos desde un archivo de texto”, utilizando como separador el espacio que separa las páginas web de sus tiempos de carga. Mediante una sencilla fórmula de “BUSCARV” hemos comprobado que realiza medidas de todas las páginas antes de copiar todos los datos a su hoja correspondiente.

Una vez teníamos todos los datos, hemos calculado la media aritmética de las diez pruebas realizadas por cada página web en cada escenario, junto con la varianza y el intervalo de confianza para asegurarnos de que los datos estaban dentro de un rango aceptable.

9. Evaluación de los protocolos

En este proyecto, vamos a evaluar el protocolo HTTP/2 y el protocolo en desarrollo QUIC en cincuenta y tres páginas web, con todos los recursos que incluyen (scripts, imágenes, gifs, css…), pertenecientes a Google Inc. (de gran variedad tanto en contenido como en peso total de la página) y sobre dieciséis escenarios de red diferentes, variando en términos de ancho de banda, tiempo de latencia y porcentaje de pérdidas, llegando a realizar un total de 16.960 pruebas.

El motivo para probar los protocolos sobre páginas de características distintas y con varios perfiles de red viene dado a que queremos que los resultados de este estudio consigan asimilarse con la mayor exactitud posible a los que tendrían ahora mismo sobre el espectro total de páginas web en internet. Debido a que puedes realizar peticiones de páginas desde diferentes redes (redes móviles, redes con fibra óptica, redes más lentas…), y no todas las páginas son iguales, consideramos que era necesario probar todos los casos posibles que estaban en nuestro alcance.

9.1. Rangos del estudio

Clasificamos las páginas en función de:

- El peso de la página: En este estudio el peso total de la página oscila entre 305 KB y 12,5 MB. Tendremos en cuenta cómo afecta esto al rendimiento de ambos protocolos y qué tipo de páginas son las más óptimas para cada uno.
- El contenido de la página: Esto lo mediremos en base a la cantidad de peticiones que se realizan para cargar todo el contenido de la página. Las páginas web evaluadas realizan entre 5 y 382 peticiones.

A su vez, configuramos los dieciséis escenarios de red como:

- Red 1: Red con una velocidad de subida y bajada de 614 kb/s (0,6 Mb/s), tiempo de latencia de 30 ms y sin pérdidas.
- Red 2: Red con una velocidad de subida y bajada de 614 kb/s (0,6 Mb/s), tiempo de latencia de 30 ms y con unas pérdidas del 0,1% de los paquetes.
- Red 3: Red con una velocidad de subida y bajada de 614 kb/s (0,6 Mb/s), tiempo de latencia de 30 ms y con unas pérdidas del 1% de los paquetes.
- Red 4: Red con una velocidad de subida y bajada de 614 kb/s (0,6 Mb/s), tiempo de latencia de 30 ms y con unas pérdidas del 10% de los paquetes.
- Red 5: Red con una velocidad de subida y bajada de 614 kb/s (0,6 Mb/s), tiempo de latencia de 270 ms y sin pérdidas.
- Red 6: Red con una velocidad de subida y bajada de 614 kb/s (0,6 Mb/s), tiempo de latencia de 270 ms y con unas pérdidas del 0,1% de los paquetes.

- Red 7: Red con una velocidad de subida y bajada de 614 kb/s (0,6 Mb/s), tiempo de latencia de 270 ms y con unas pérdidas del 1% de los paquetes.
- Red 8: Red con una velocidad de subida y bajada de 614 kb/s (0,6 Mb/s), tiempo de latencia de 270 ms y con unas pérdidas del 10% de los paquetes.
- Red 9: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 30 ms y sin pérdidas.
- Red 10: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 30 ms y con unas pérdidas del 0,1% de los paquetes.
- Red 11: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 30 ms y con unas pérdidas del 1% de los paquetes.
- Red 12: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 30 ms y con unas pérdidas del 10% de los paquetes.
- Red 13: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 270 ms y sin pérdidas.
- Red 14: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 270 ms y con unas pérdidas del 0,1% de los paquetes.
- Red 15: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 270 ms y con unas pérdidas del 1% de los paquetes.
- Red 16: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 270 ms y con unas pérdidas del 10% de los paquetes.

	Mínimo	Máximo
Peso de la página	305 KB	12,5 MB
Número de peticiones	5	382
Velocidad de subida y bajada	0,6 Mb/s	12 Mb/s
Tiempo de latencia	30 ms	270 ms
Porcentaje de pérdidas	0%	10%

Tabla 2 - Rangos de las pruebas del estudio

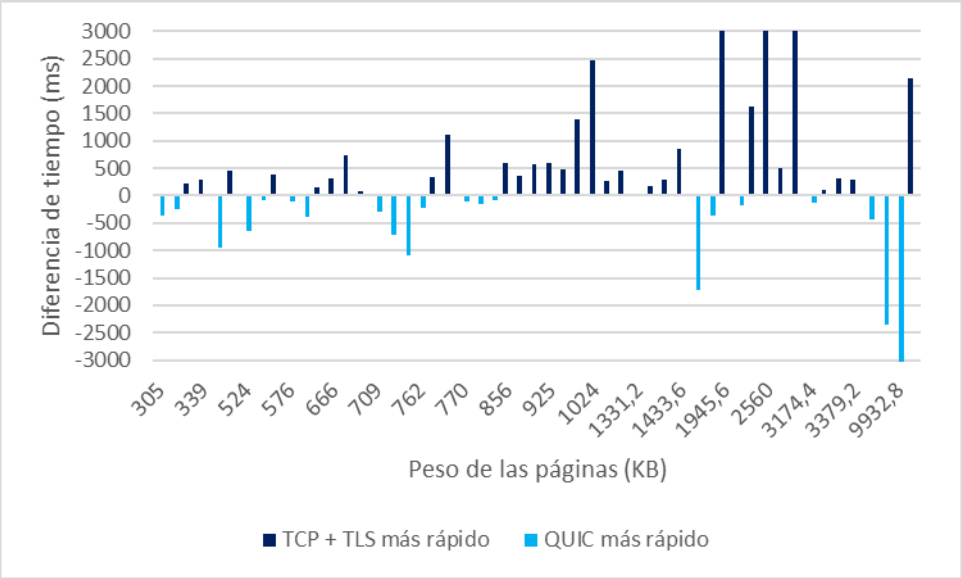
9.2. Resultados

En este apartado analizaremos los resultados obtenidos tras realizar las pruebas, midiendo para ello tres variables:

1. La diferencia de tiempo entre la media aritmética de las diez pruebas de cada página realizadas de cada protocolo. Esta medida se representa en la primera (en función del peso) y segunda gráfica (en función del número de peticiones) de cada apartado.

2. La varianza entre las diez medidas de cada página en cada protocolo. Aquí comprobamos los picos en los que la red ha ido más lenta y las medidas son menos fiables. Esta medida está representada en la tercera gráfica de cada apartado.
3. El intervalo de confianza de las diez medidas de cada página en cada protocolo. Para calcularlo hemos tomado un nivel de confianza del 90% (un valor alfa de 0,1). Esta medida nos proporciona comprobar que los resultados no disten mucho entre sí y estén en un rango adecuado. Podemos encontrar su representación en la cuarta gráfica de cada apartado.

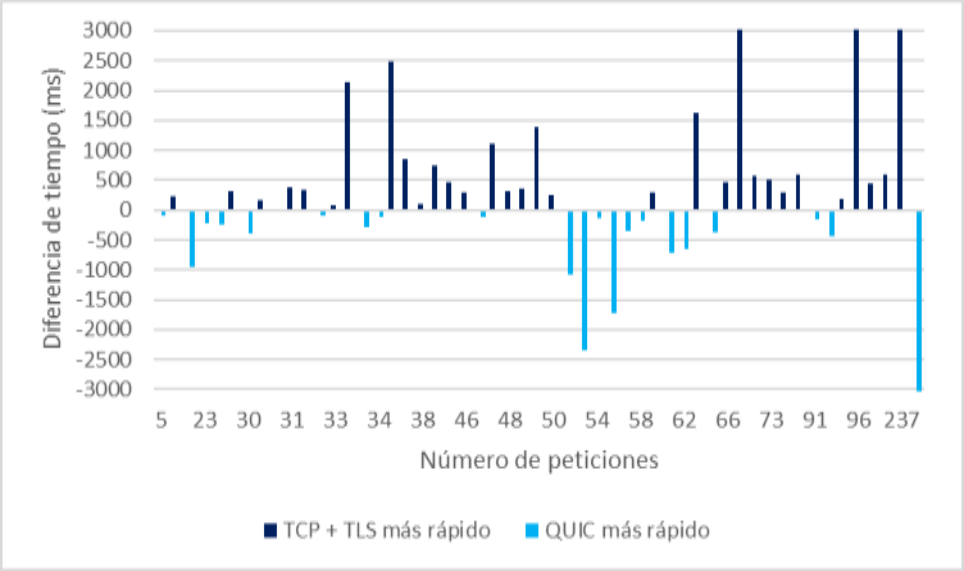
9.2.1 Red 1: Red con una velocidad de subida y bajada de 0,6 Mb/s, tiempo de latencia de 30 ms y sin pérdidas.



Gráfica 1 - Diferencia de tiempos en función del peso para la Red 1

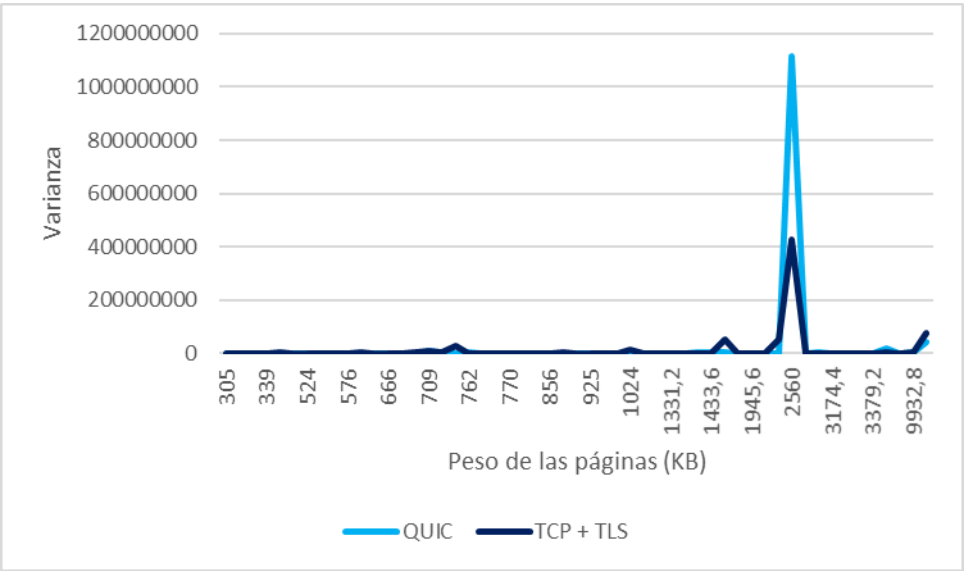
Como podemos comprobar, bajo esta configuración la combinación de TCP y TLS suele ser más rápida en un mayor número de páginas.

Además, si ordenamos las páginas en función del número de peticiones, observamos que TCP y TLS tiene un mejor rendimiento sobre páginas con mayor número de peticiones.



Gráfica 2 - Diferencia de tiempos en función del número de peticiones para la Red 1

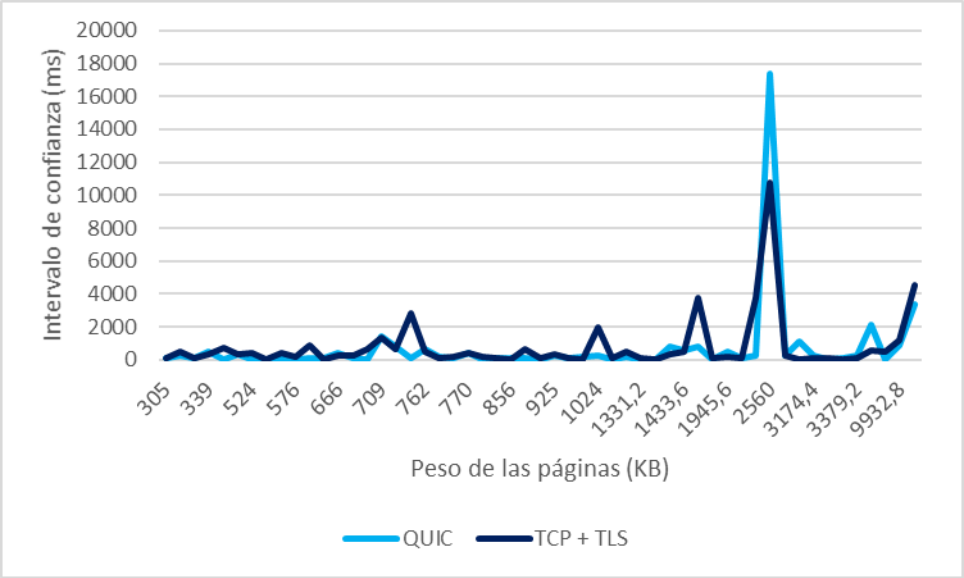
Cuando observamos la varianza, nos encontramos con la siguiente situación:



Gráfica 3 - Varianza en la Red 1

Como podemos observar, a lo largo de los tres días que se ha estado ejecutando el *script* para TCP y TLS, se han observado diversos picos en la red que han incrementado la varianza en cuatro puntos, mientras que en el caso de QUIC sólo ha existido un pico de red, pero más pronunciado. Por otro lado, también observamos un aumento de la varianza cuanto más pesadas sean las páginas web.

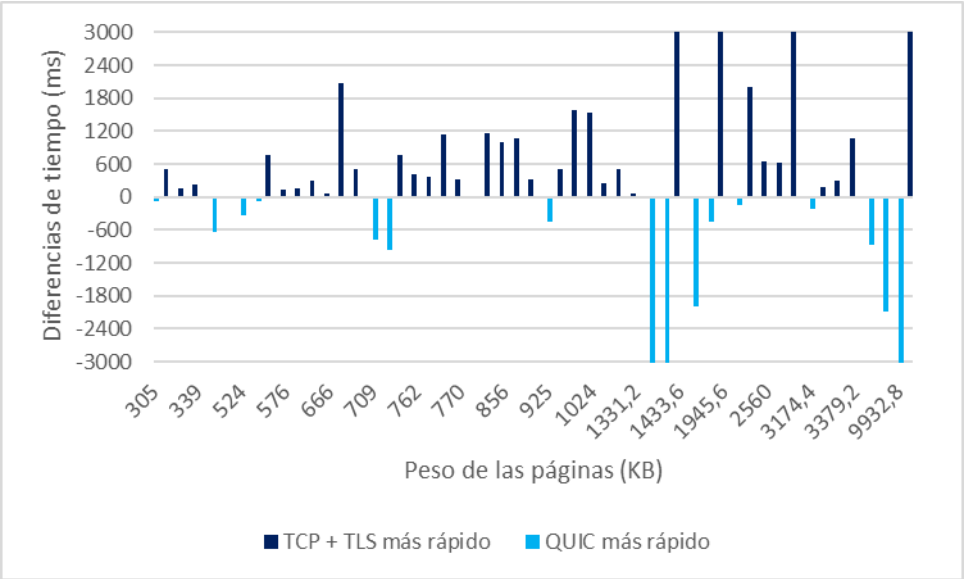
Observando el intervalo de confianza, obtenemos la siguiente gráfica:



Gráfica 4 - Intervalo de confianza en la Red 1

Observamos que el intervalo de confianza suele estar en niveles aceptables (menos de dos segundos de margen) salvo en los picos de red mencionados previamente, por lo que consideramos los resultados obtenidos como válidos.

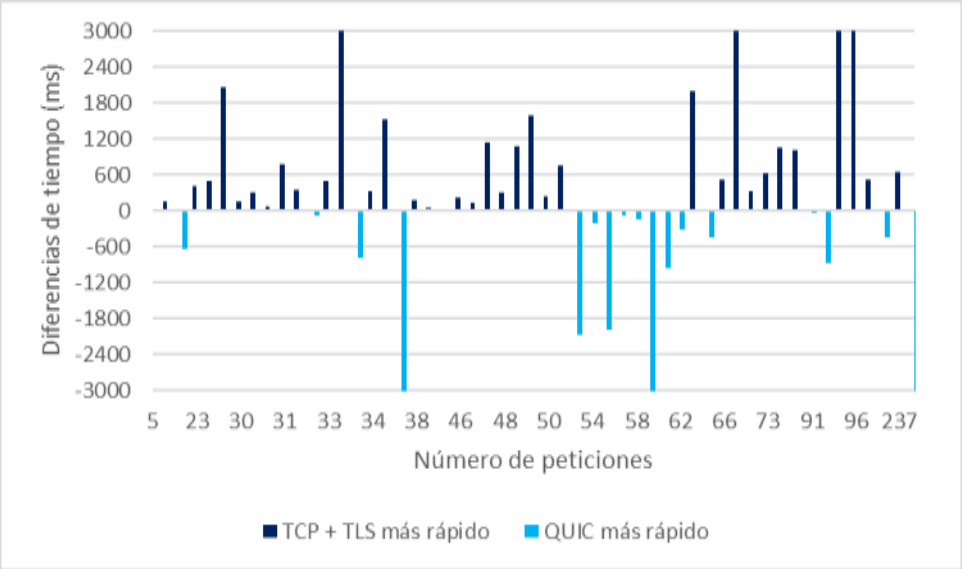
9.2.2 Red 2: Red con una velocidad de subida y bajada de 0,6 Mb/s, tiempo de latencia de 30 ms y con unas pérdidas del 0,1% de los paquetes.



Gráfica 5 - Diferencia de tiempos en función del peso para la Red 2

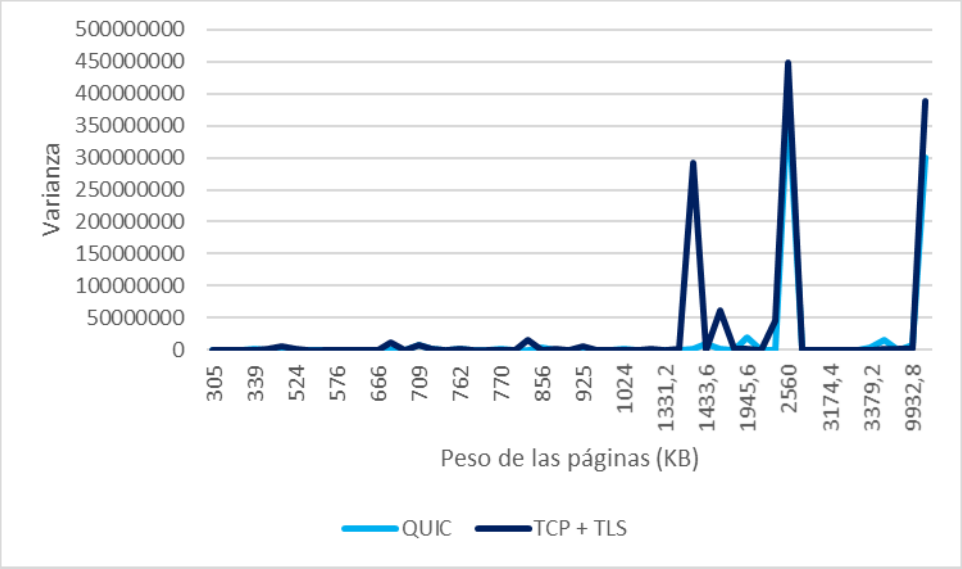
Como podemos comprobar, pese al aumento de las pérdidas el módulo FEC de QUIC no consigue compensar la diferencia de tiempo con el uso de los protocolos TCP y TLS. En

este caso vemos que también se mantiene la predominancia de TCP y TLS sobre páginas con un mayor número de peticiones.



Gráfica 6 - Diferencia de tiempos en función del número de peticiones para la Red 2

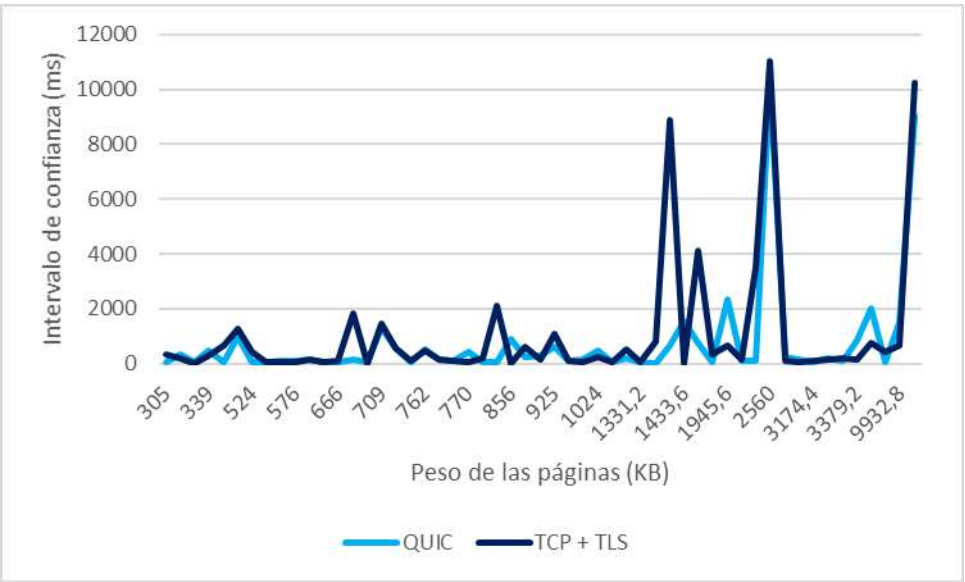
Cuando observamos la varianza, nos encontramos con la siguiente situación:



Gráfica 7 - Varianza en la Red 2

En esta ocasión podemos observar tres picos en la red (coincidiendo el más pronunciado con el analizado previamente).

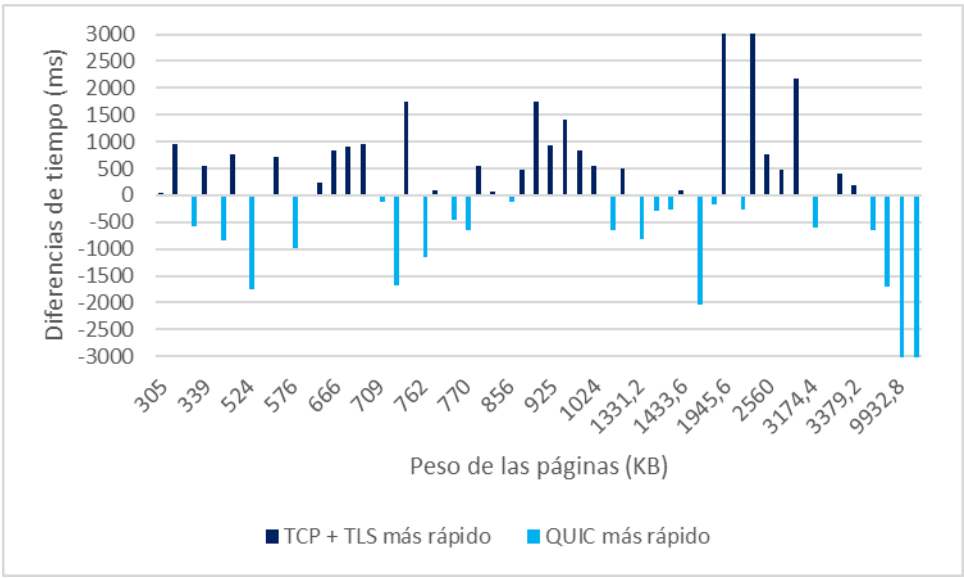
Analizando el intervalo de confianza, llegamos a la siguiente conclusión:



Gráfica 8 - Intervalo de confianza en la Red 2

Aunque a priori parezca una situación más inestable, los picos se dan para unos valores más reducidos a la ocasión anterior, siendo el máximo un valor de once segundos que coincide con el pico de red mencionado previamente.

9.2.3 Red 3: Red con una velocidad de subida y bajada de 0,6 Mb/s, tiempo de latencia de 30 ms y con unas pérdidas del 1% de los paquetes.

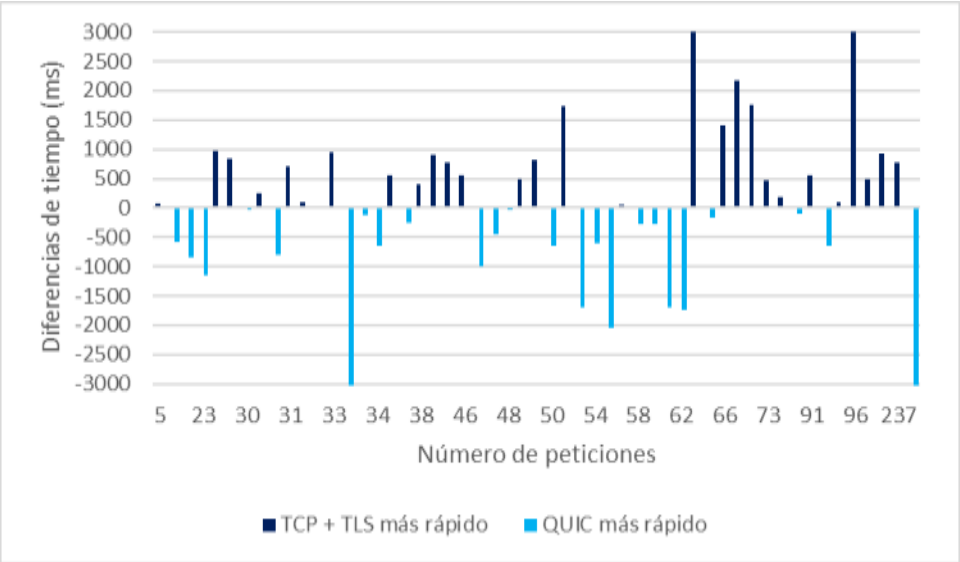


Gráfica 9 - Diferencia de tiempos en función del peso para la Red 3

Tras analizar la diferencia de tiempos, nos encontramos que el módulo FEC de QUIC empieza a afectar a la velocidad del protocolo, haciendo que un mayor número de

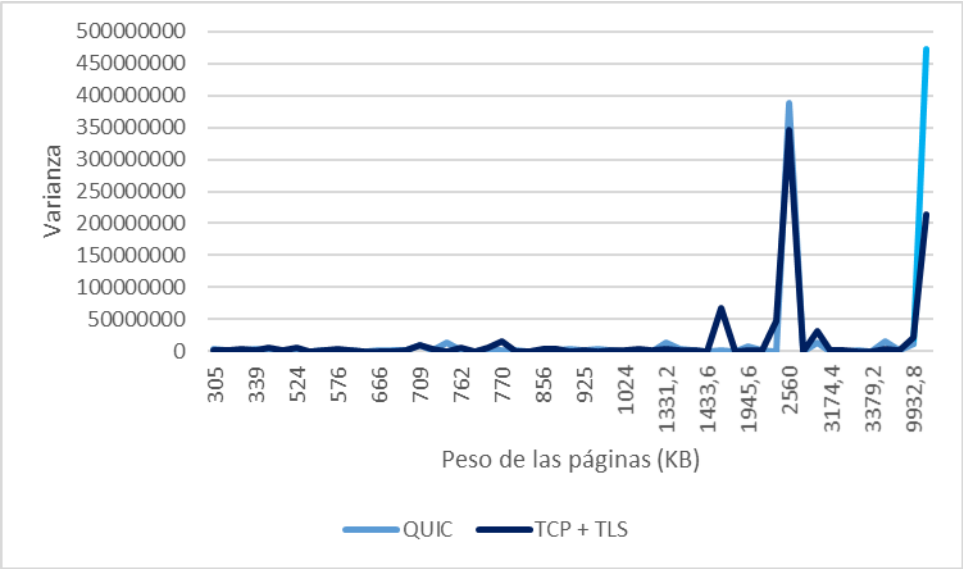
páginas sean más rápidas que TCP y TLS y que aquellas páginas en las que TCP y TLS siguen superando a QUIC, lo hagan con menor diferencia de tiempo.

TCP y TLS sigue obteniendo mayor rendimiento en las páginas con mayor número de peticiones:

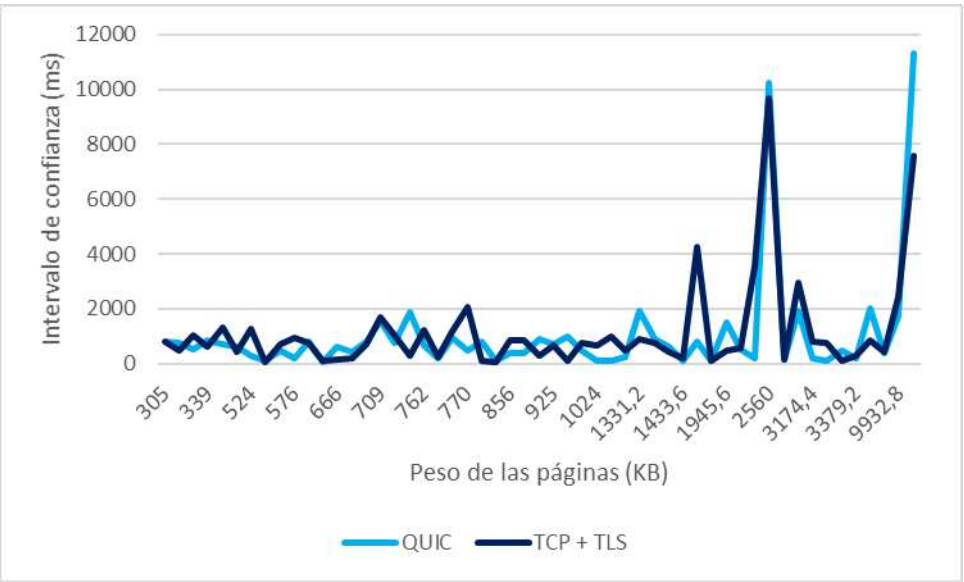


Gráfica 10 - Diferencia de tiempos en función del número de peticiones para la Red 3

Respecto a la varianza e intervalo de confianza, comprobamos que la situación se mantiene estable respecto a la situación anterior:

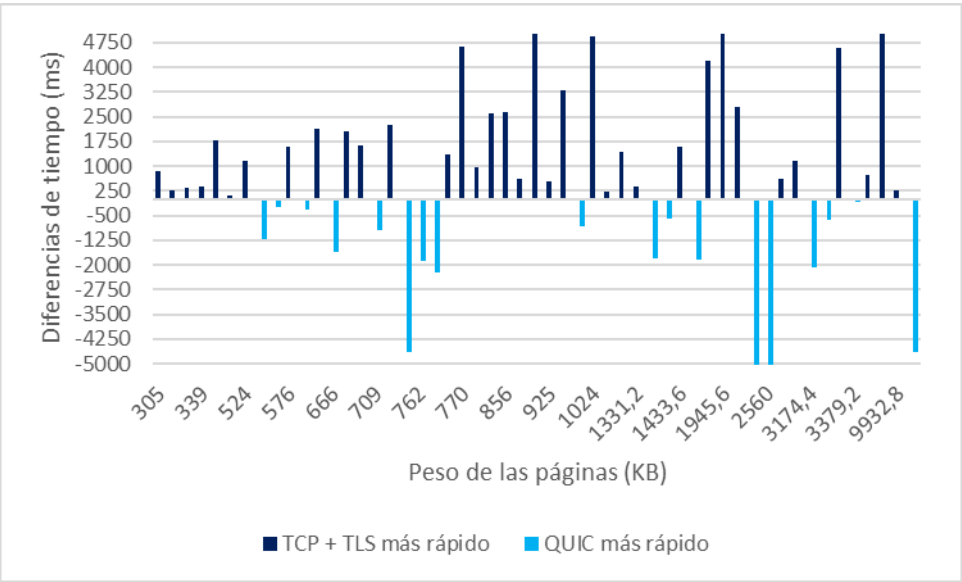


Gráfica 11 - Varianza en la Red 3



Gráfica 12 - Intervalo de confianza en la Red 3

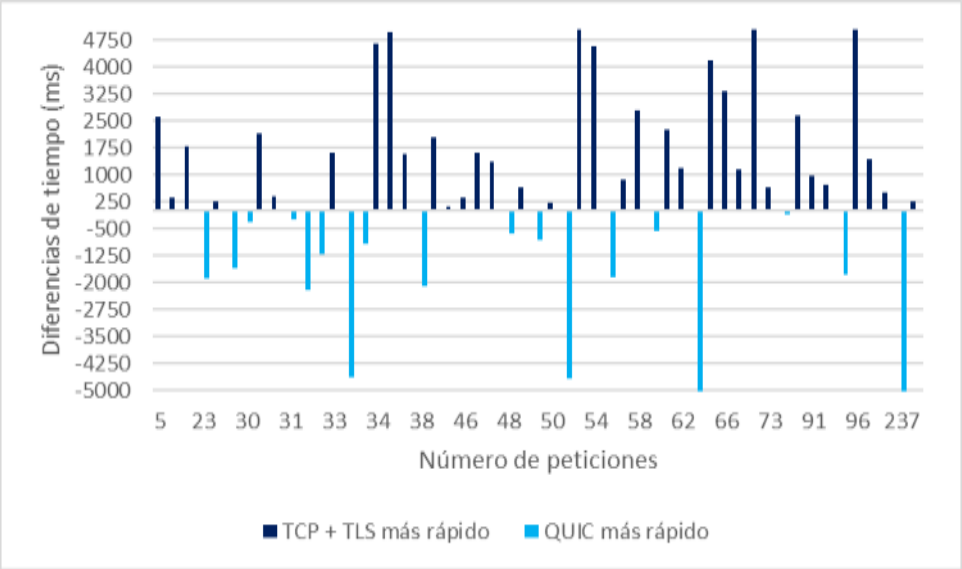
9.2.4 Red 4: Red con una velocidad de subida y bajada de 0,6 Mb/s, tiempo de latencia de 30 ms y con unas pérdidas del 10% de los paquetes.



Gráfica 13 - Diferencia de tiempos en función del peso para la Red 4

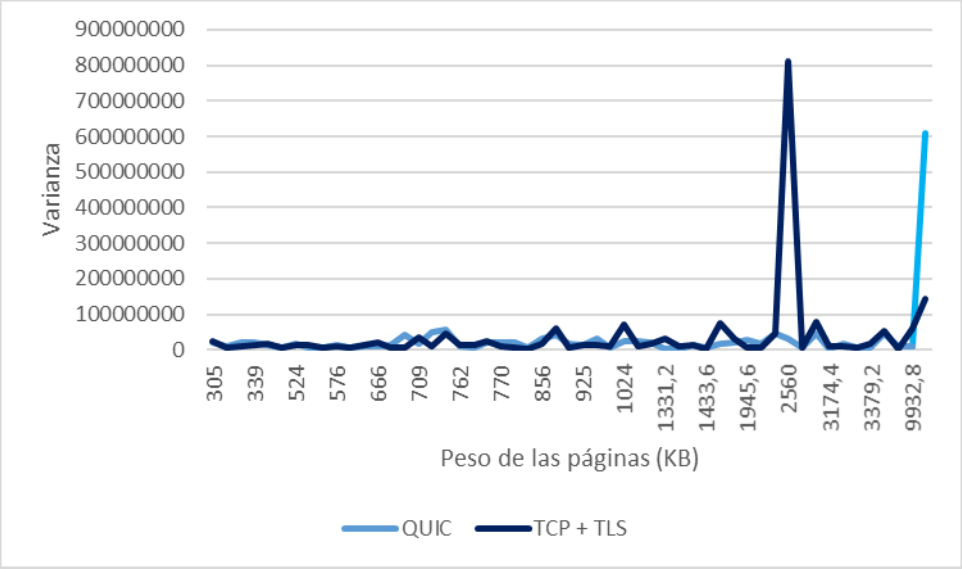
Observando las diferencias de tiempo, vemos que el rendimiento de QUIC ha empeorado respecto al escenario anterior. Esto probablemente se deba a que el módulo FEC que implementa QUIC no es capaz de soportar un porcentaje tan alto de pérdidas por lo que TCP y TLS vuelven a superar a este protocolo.

TCP y TLS siguen obteniendo mejor rendimiento sobre páginas con mayor número de peticiones:

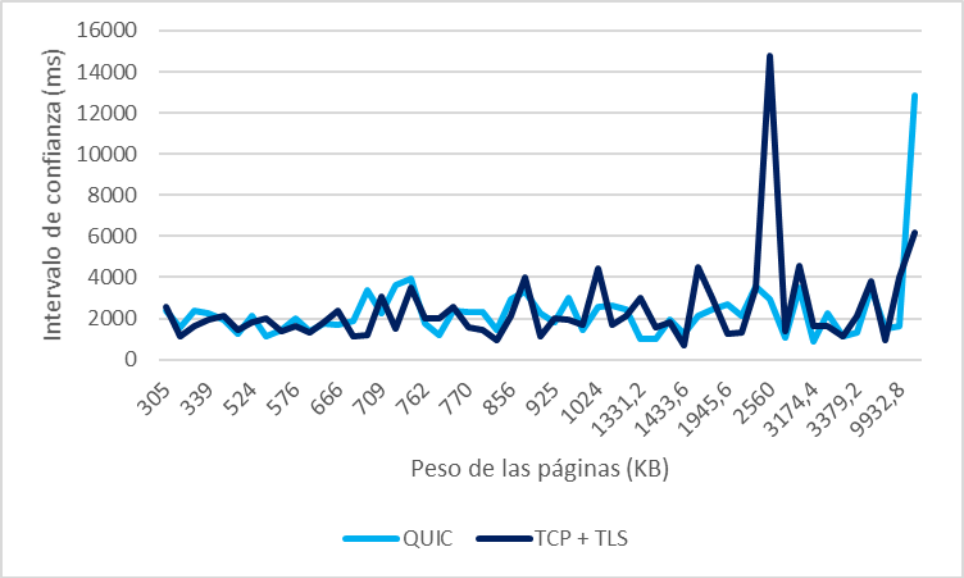


Gráfica 14 - Diferencia de tiempos en función del número de peticiones para la Red 4

Observando la varianza y los valores de confianza, observamos que la situación sigue estable:

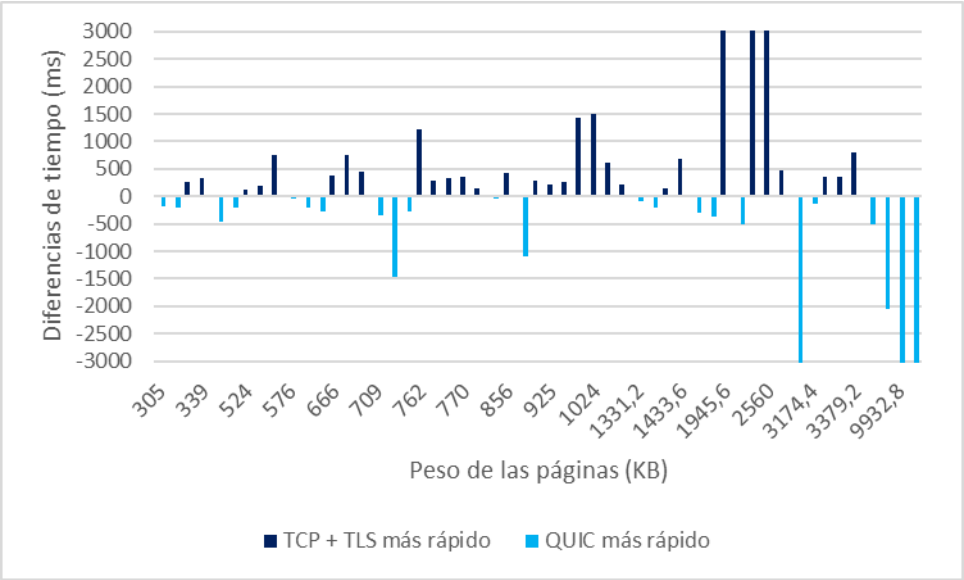


Gráfica 15 - Varianza en la Red 4



Gráfica 16 - Intervalo de confianza en la Red 4

9.2.5 Red 5: Red con una velocidad de subida y bajada de 0,6 Mb/s, tiempo de latencia de 270 ms y sin pérdidas.

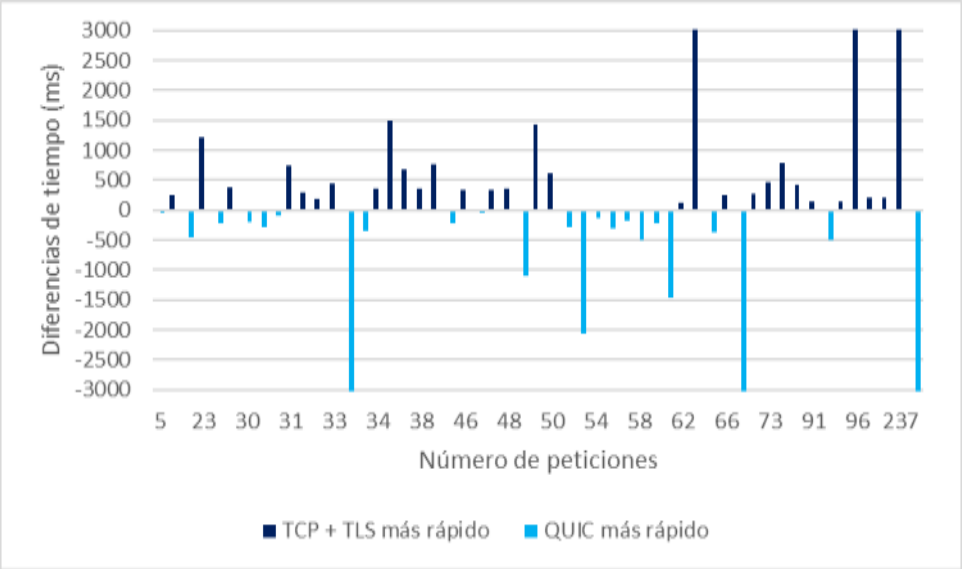


Gráfica 17 - Diferencia de tiempos en función del peso para la Red 5

En este escenario, pese a que TCP y TLS y QUIC están bastante igualados, nos interesa destacar otro aspecto importante de la comparativa: Respecto a su escenario análogo anterior, con un tiempo de latencia más bajo, QUIC aumenta su rendimiento frente a TCP y TLS.

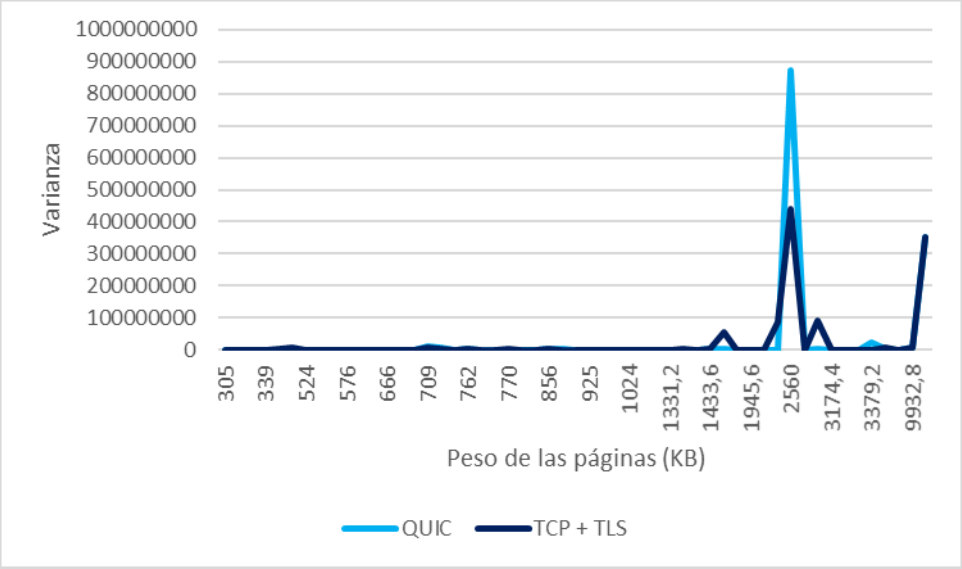
Esto es causado por el uso de UDP, que frente a retransmisiones reacciona mejor al no frenar el resto de paquetes posteriores al afectado.

TCP y TLS siguen demostrando mayor rendimiento sobre páginas con mayor número de peticiones:

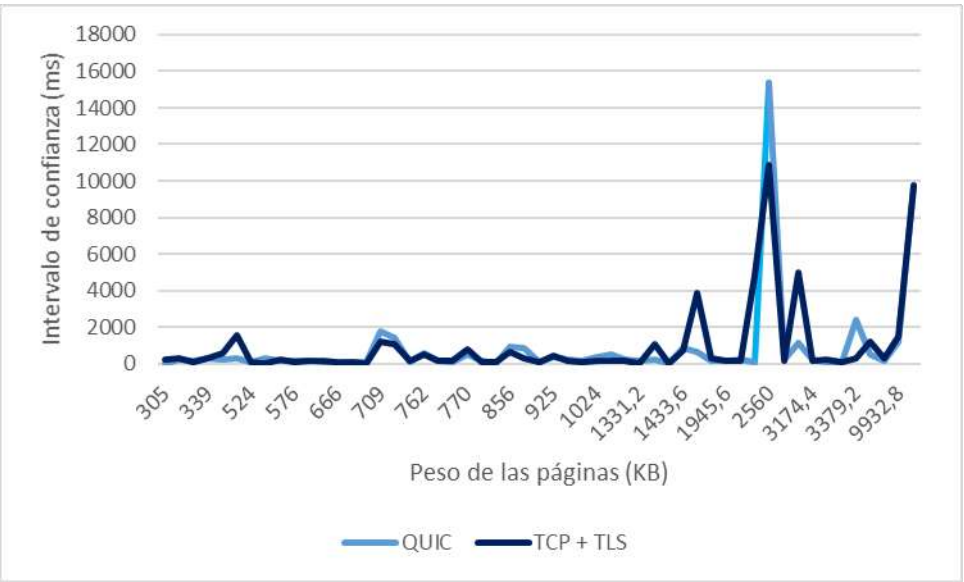


Gráfica 18 - Diferencia de tiempos en función del número de peticiones para la Red 5

Observando la varianza y el intervalo de confianza, nos damos cuenta de que ambos siguen con los picos comentados previamente, aunque su valor ha disminuido respecto al escenario con treinta milisegundos de latencia.

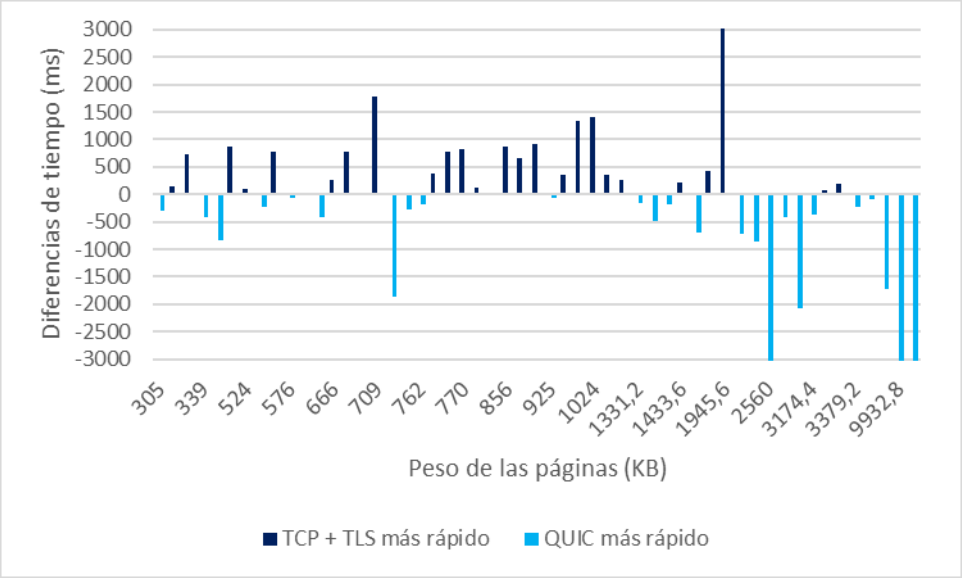


Gráfica 19 - Varianza en la Red 5



Gráfica 20 - Intervalo de confianza en la Red 5

9.2.6 Red 6: Red con una velocidad de subida y bajada de 0,6 Mb/s, tiempo de latencia de 270 ms y con unas pérdidas del 0,1% de los paquetes.

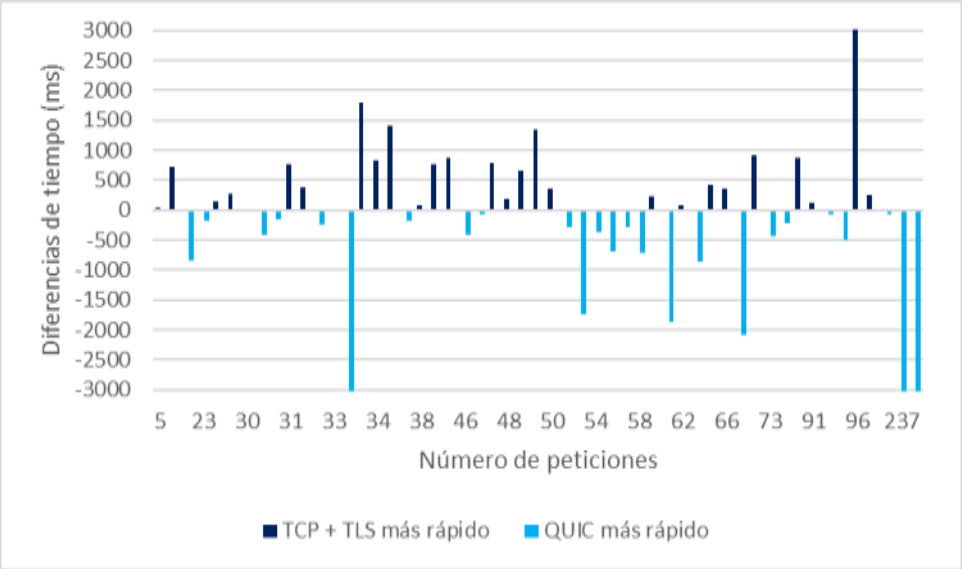


Gráfica 21 - Diferencia de tiempos en función del peso para la Red 6

Analizando las diferencias de tiempo llegamos a la conclusión de que no solo mejora respecto al escenario de la Red 5, sino que además ha mejorado respecto al de la Red 2, como esperábamos debido al comportamiento explicado en el apartado anterior.

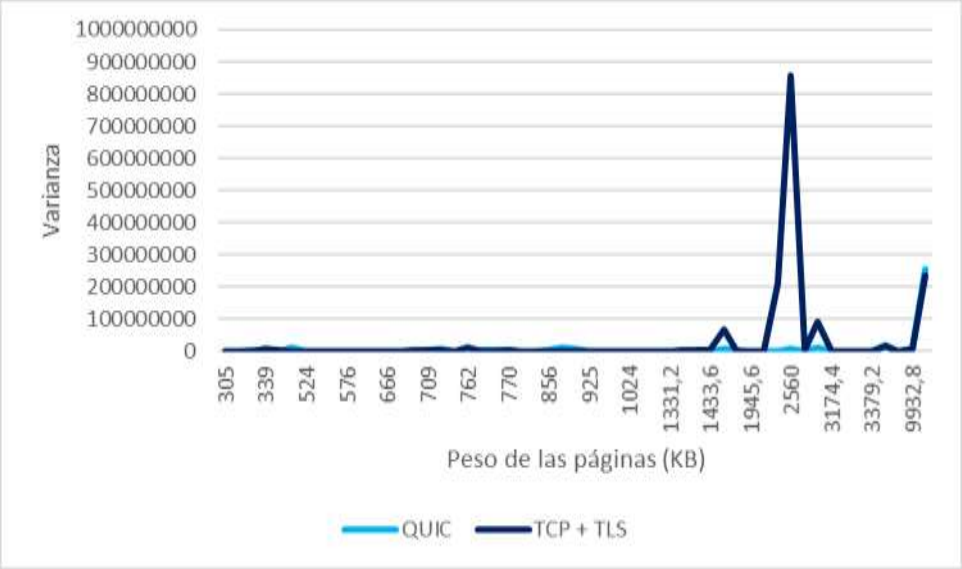
Además, comprobamos que el rendimiento de QUIC se acentúa en las páginas de mayor peso, superando con diferencia a TCP y TLS salvo en dos casos en los que casi lo iguala.

En este caso no observamos ninguna diferencia apreciable entre TCP y TLS y QUIC en cuanto a páginas con el mismo número de peticiones:

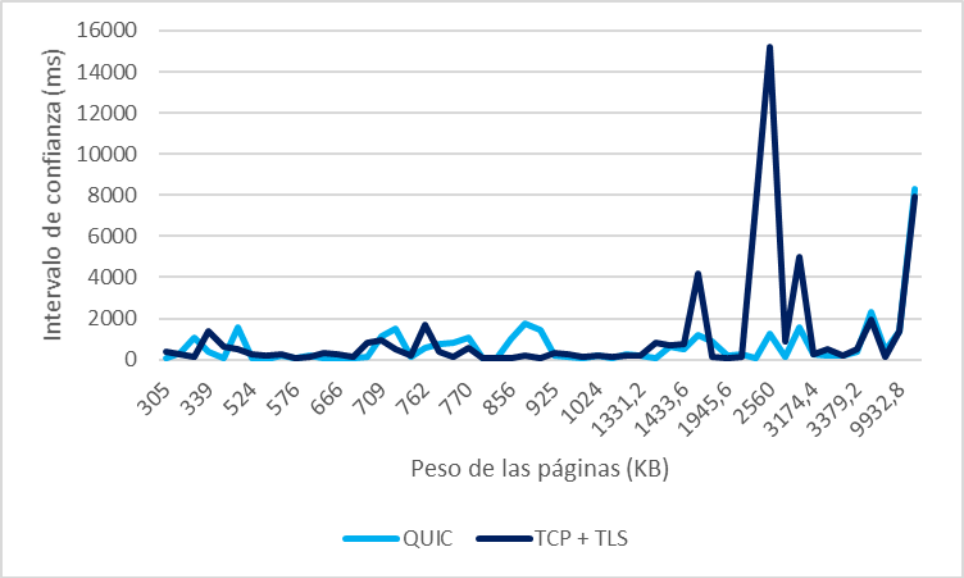


Gráfica 22 - Diferencia de tiempos en función del número de peticiones para la Red

Analizando la varianza e intervalo de confianza, comprobamos que el pico de red de QUIC ha desaparecido, aunque aún se mantiene el de TCP y TLS:

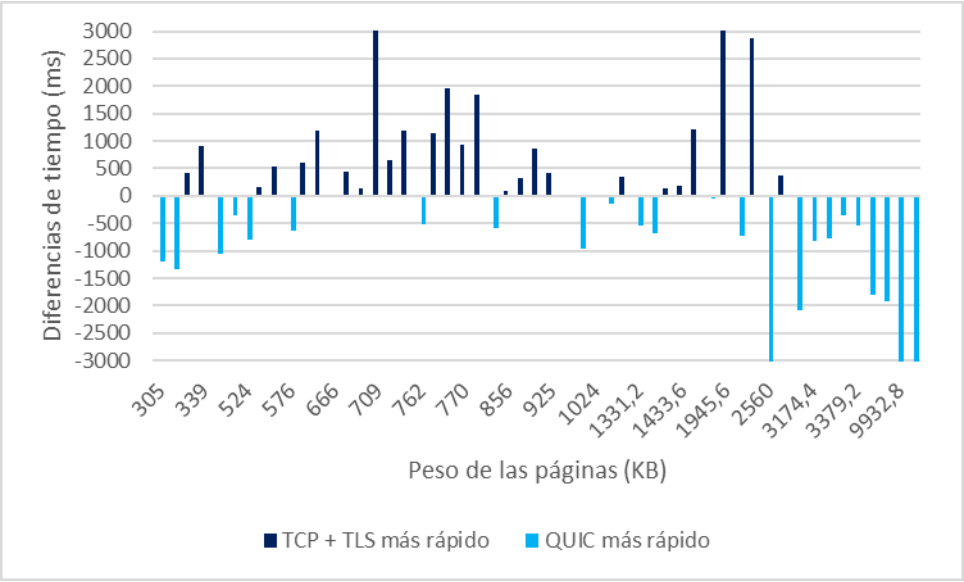


Gráfica 23 - Varianza en la Red 6



Gráfica 24 - Intervalo de confianza en la Red 6

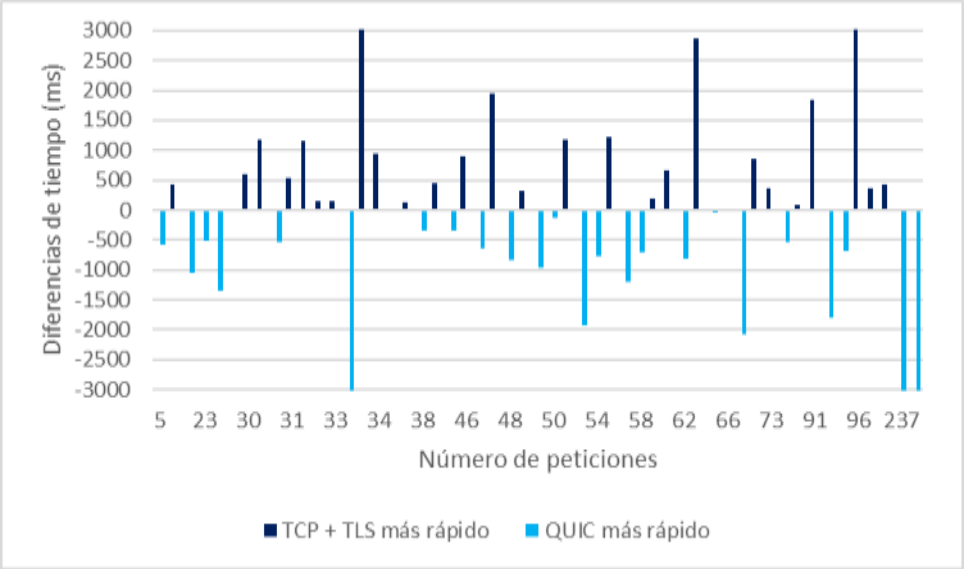
9.2.7 Red 7: Red con una velocidad de subida y bajada de 0,6 Mb/s, tiempo de latencia de 270 ms y con unas pérdidas del 1% de los paquetes.



Gráfica 25 - Diferencia de tiempos en función del peso para la Red 7

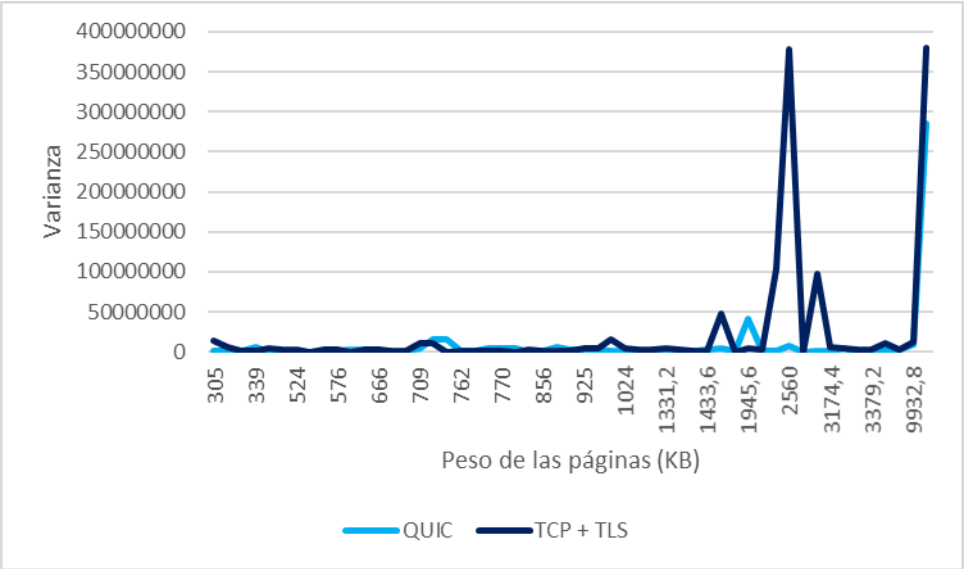
En este escenario comprobamos que QUIC sigue superando a TCP y TLS tanto en general como específicamente en las páginas con mayor peso.

En este escenario tampoco observamos ninguna diferencia apreciable basándonos en el número de peticiones de las páginas web:

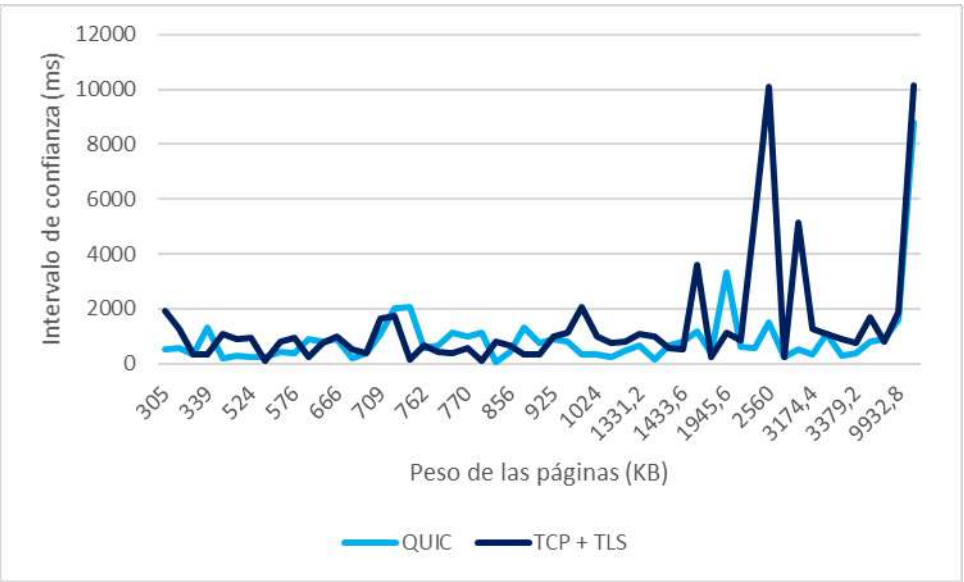


Gráfica 26 - Diferencia de tiempos en función del número de peticiones para la Red 7

Además, el escenario sigue mostrando el pico de red en TCP y TLS tanto en la varianza como en el intervalo de confianza:

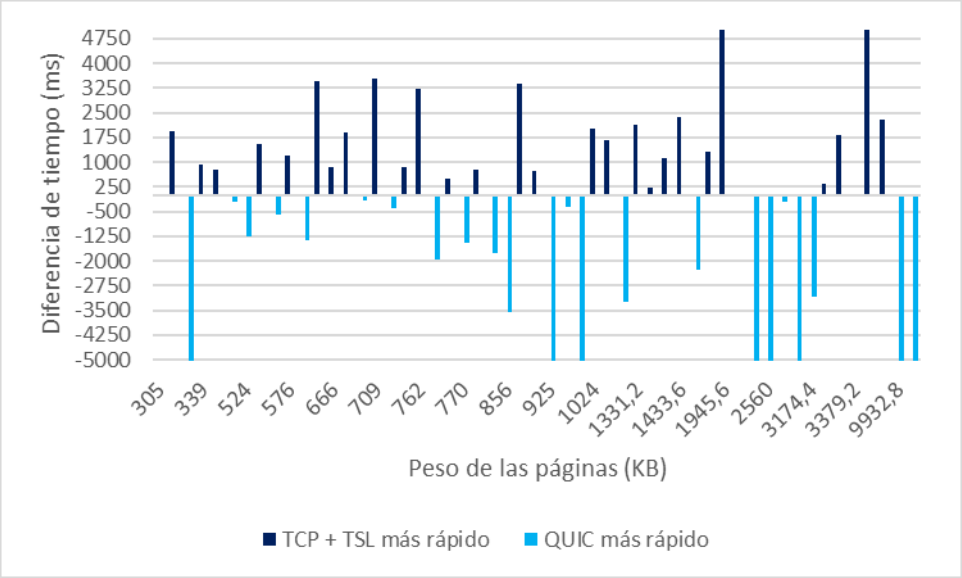


Gráfica 27 -Varianza en la Red 7



Gráfica 28 -Intervalo de confianza en la Red 7

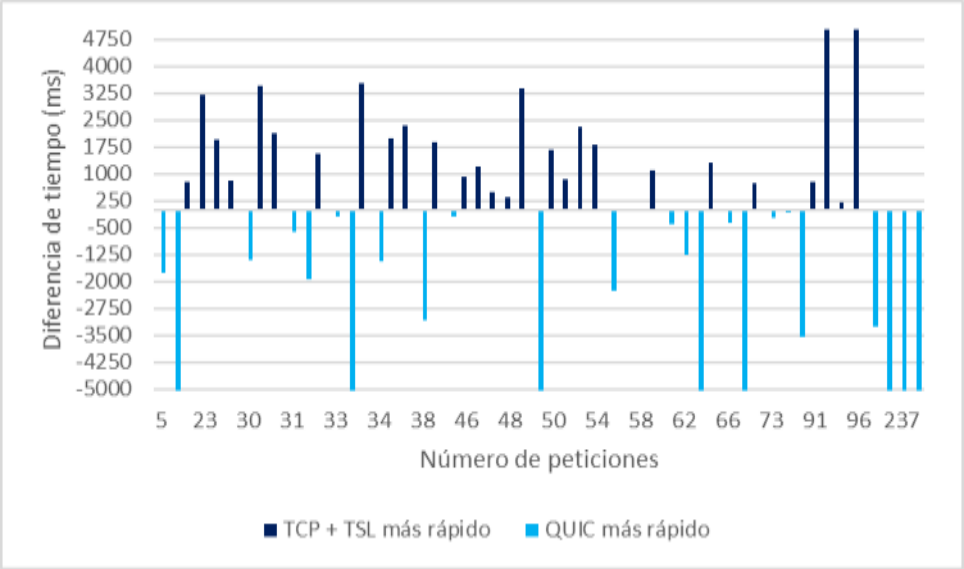
9.2.8 Red 8: Red con una velocidad de subida y bajada de 0,6 Mb/s, tiempo de latencia de 270 ms y con unas pérdidas del 10% de los paquetes.



Gráfica 29 - Diferencia de tiempos en función del peso para la Red 8

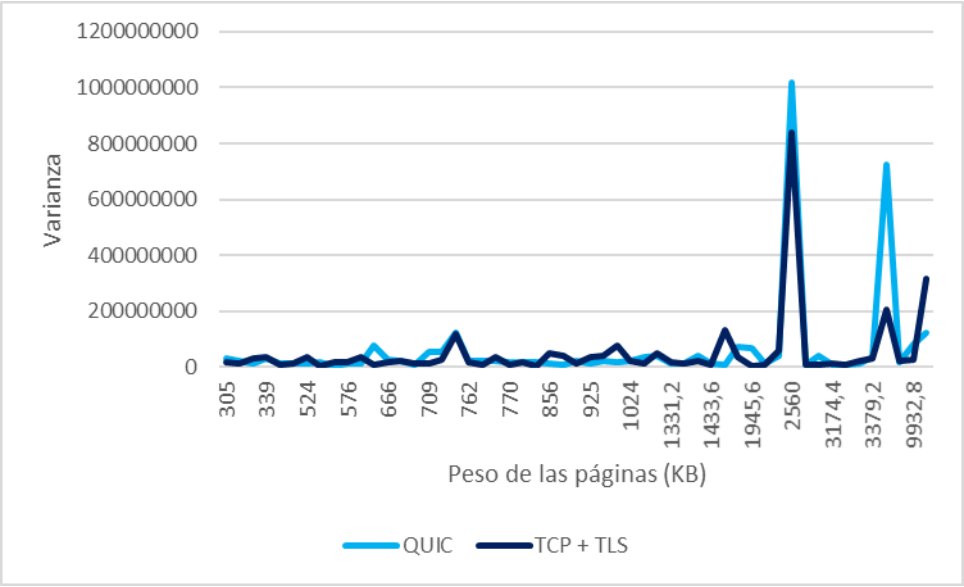
Una vez más, notamos mejora respecto al mismo escenario con una latencia menor, aunque no muestra un rendimiento mayor a TCP y TLS, sino que el rendimiento está bastante igualado.

En este caso, QUIC supera a TCP y TLS en cuanto a tiempo de carga para páginas de mayor número de peticiones:

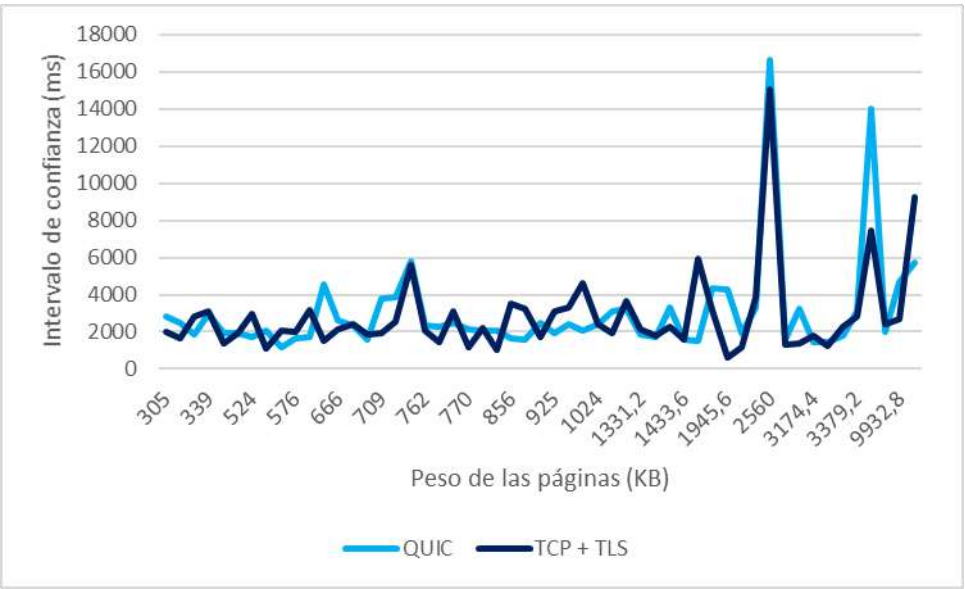


Gráfica 30 - Diferencia de tiempos en función del número de peticiones para la Red 8

Por otro lado, reaparece el pico de red para QUIC en la varianza e intervalo de confianza:

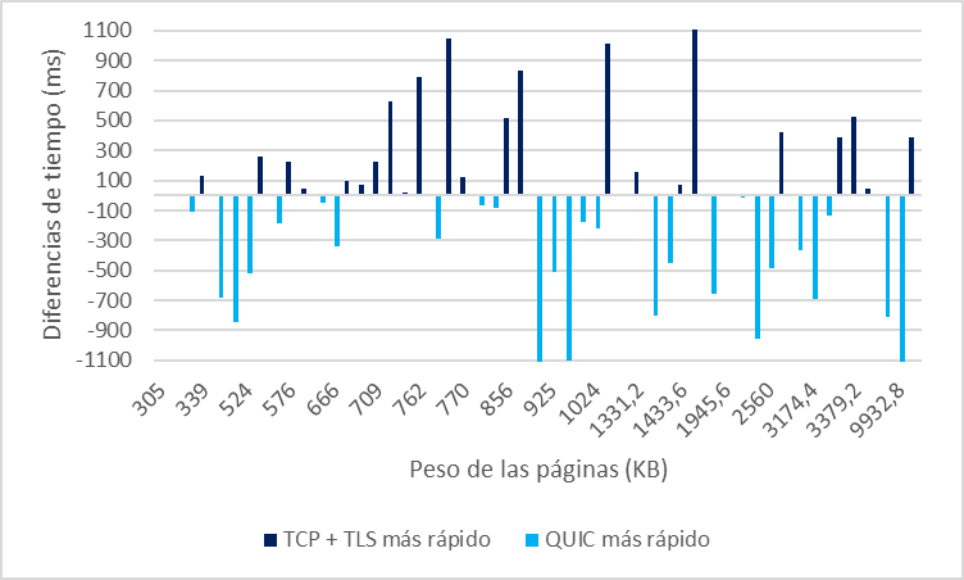


Gráfica 31 - Varianza en la Red 8



Gráfica 32 - Intervalo de confianza en la Red 8

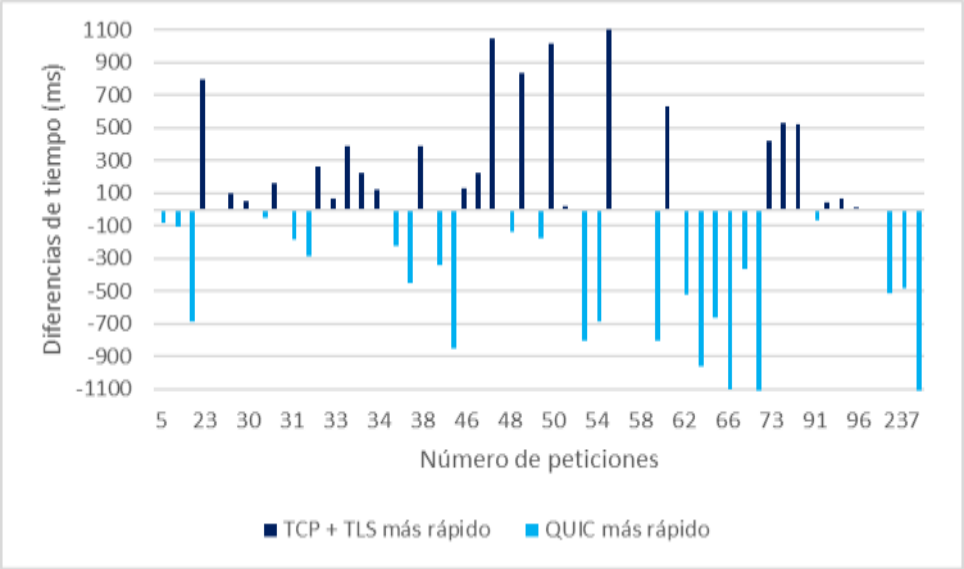
9.2.9 Red 9: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 30 ms y sin pérdidas.



Gráfica 33 - Diferencia de tiempos en función del peso para la Red 9

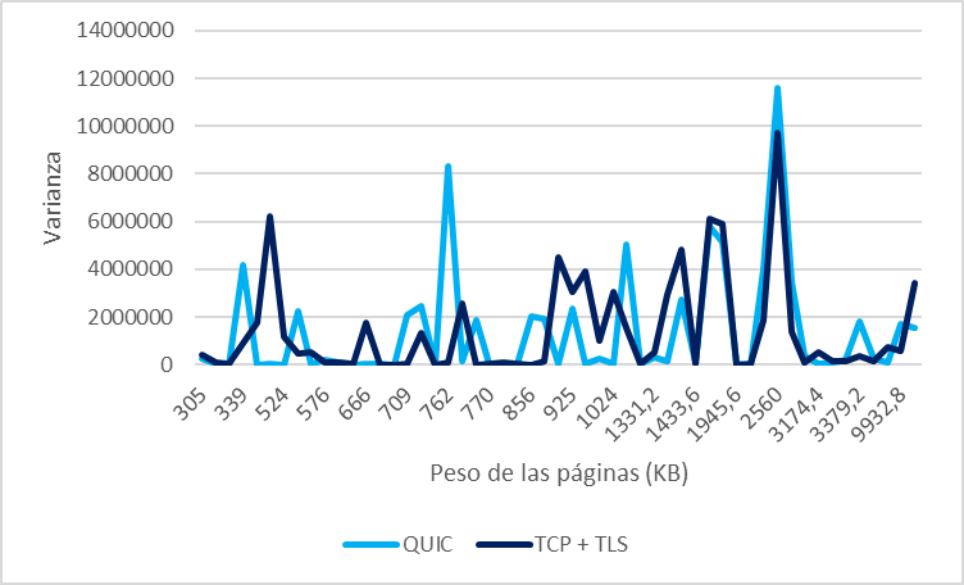
Comparando este resultado con los dos escenarios similares anteriores, comprobamos que el rendimiento de QUIC mejora bastante con el aumento de la velocidad de red, aunque no observamos ninguna diferencia apreciable entre páginas web pesadas o ligeras.

Bajo este escenario, también observamos que QUIC tiene mejor rendimiento bajo páginas con mayor número de peticiones:

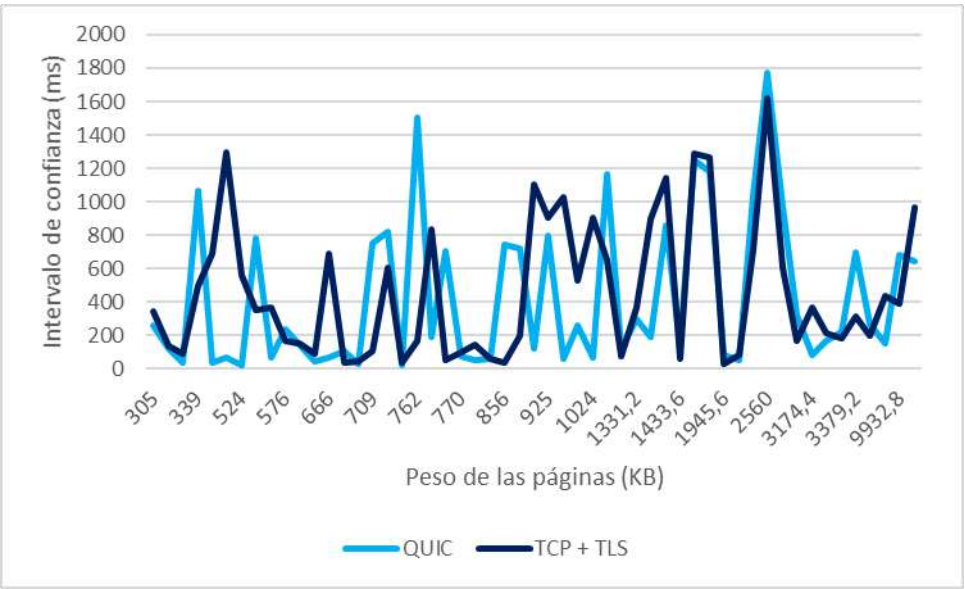


Gráfica 34 - Diferencia de tiempos en función del peso para la Red 9

Observando la varianza y el intervalo de confianza, vemos que el pico de red ya ha desaparecido en este escenario. Pese a que parezca una varianza o intervalo de confianza inestable, si nos fijamos en los valores nos daremos cuenta que son mucho menores a los hallados previamente:

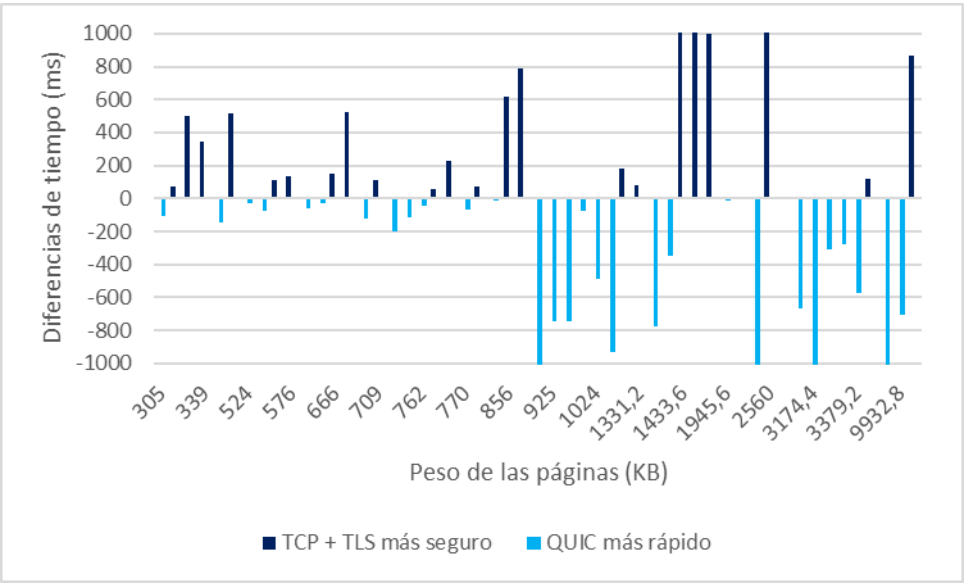


Gráfica 35 - Diferencia de tiempos en la Red 9



Gráfica 36 - Intervalo de confianza en la Red 9

9.2.10 Red 10: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 30 ms y con unas pérdidas del 0,1% de los paquetes.

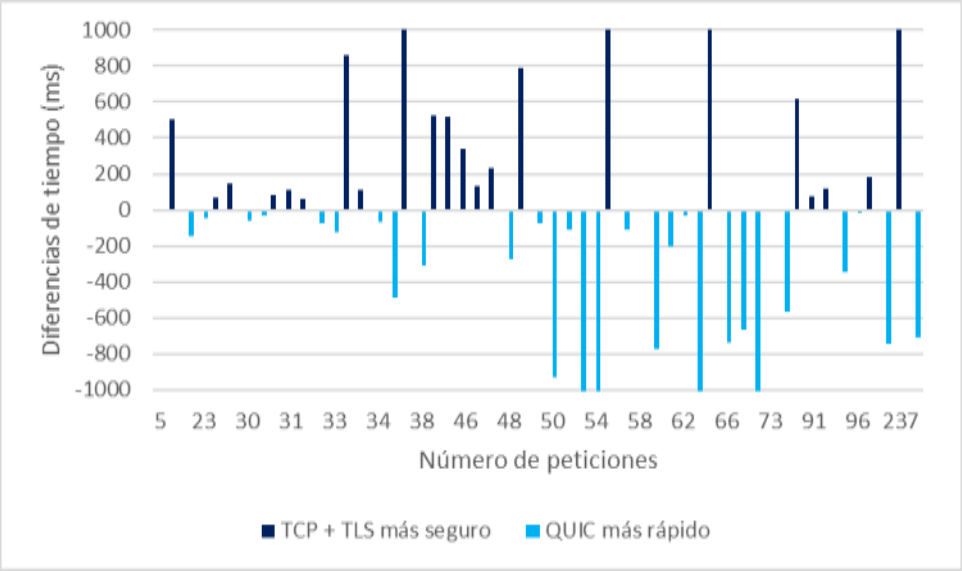


Gráfica 37 - Diferencia de tiempos en función del peso para la Red 10

Observando la gráfica, comprobamos que QUIC supera en rendimiento a TCP y TLS sobre páginas web pesadas, aunque no se detecta un aumento del rendimiento respecto al escenario sin pérdidas anterior.

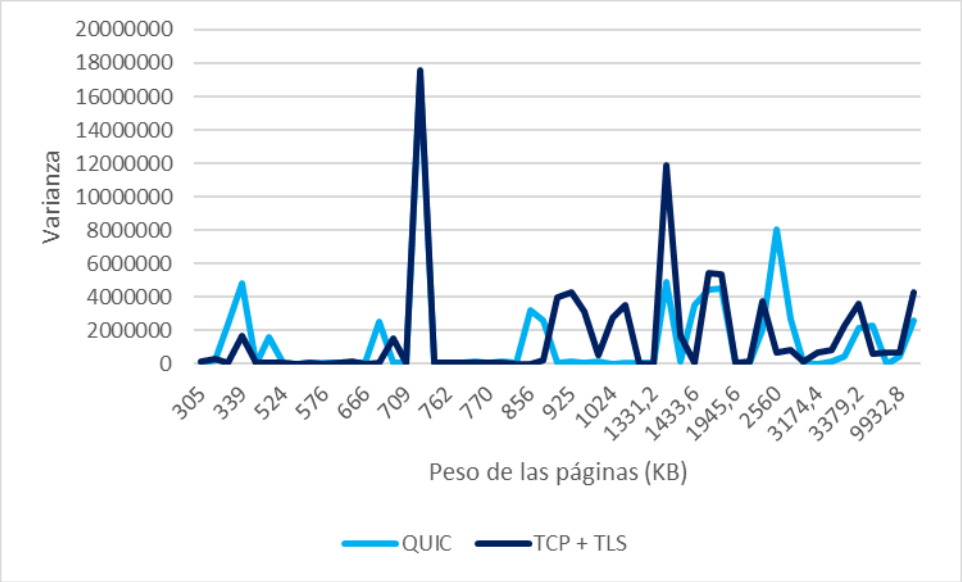
Comparándolo con los escenarios de red con menor ancho de banda, comprobamos que QUIC sigue siendo mucho más eficaz cuanto mayor sea la velocidad de red.

Comparando las medidas según el número de peticiones, observamos un mejor rendimiento del protocolo QUIC para páginas con mayor número de peticiones:

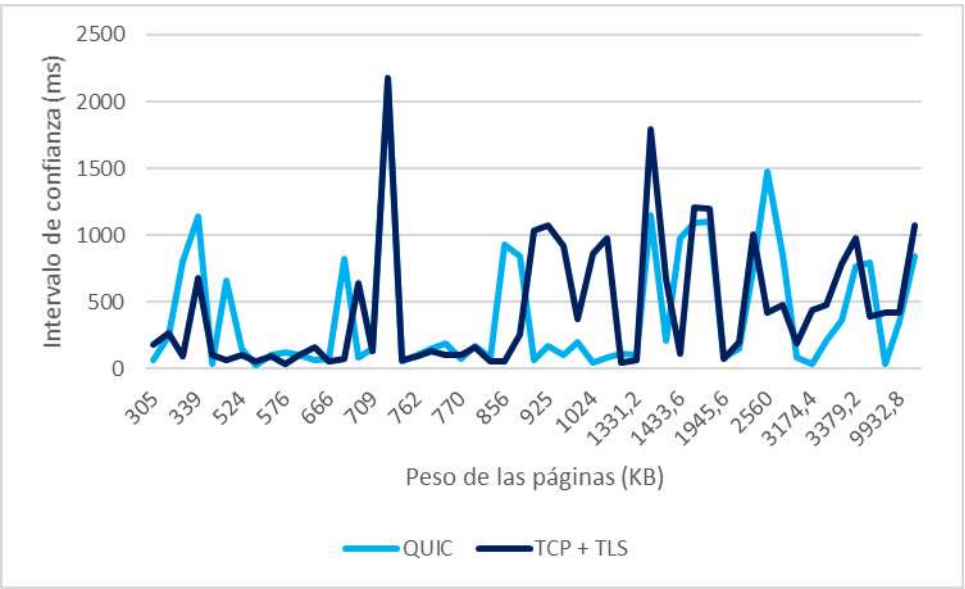


Gráfica 38 - Diferencia de tiempos en función del número de peticiones para la Red 10

Por otro lado, la varianza y el intervalo de confianza siguen en rangos aceptables:

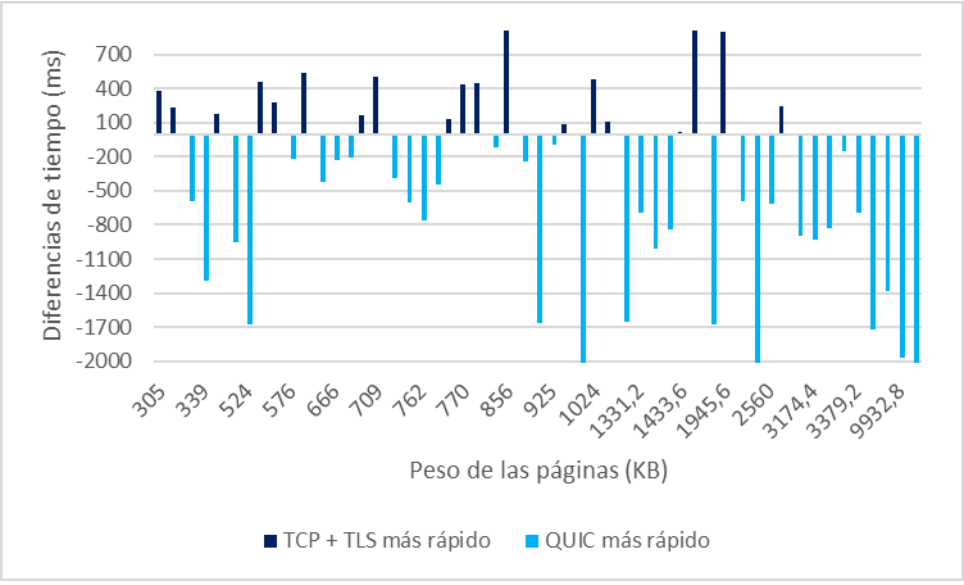


Gráfica 39 - Varianza en la Red 10



Gráfica 40 - Intervalo de confianza en la Red 10

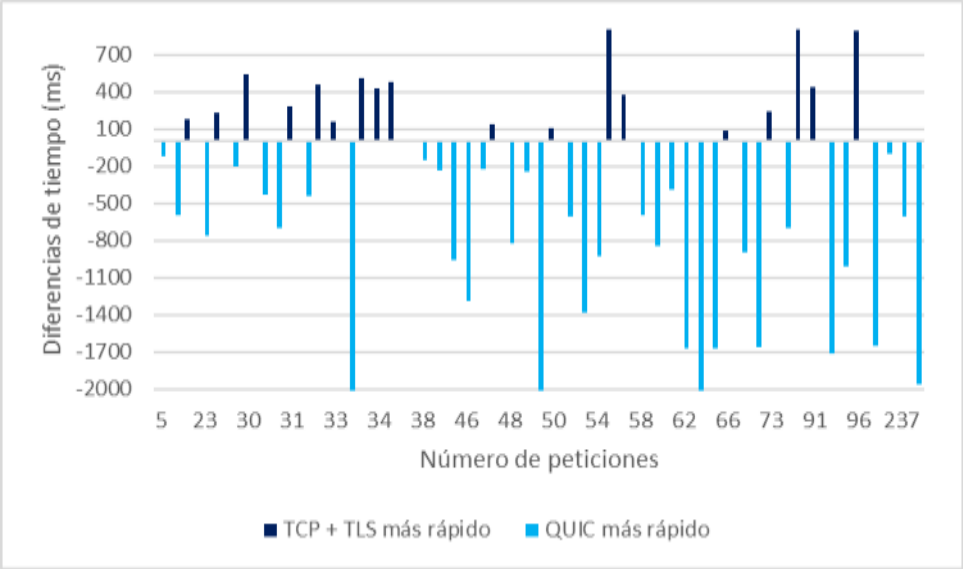
9.2.11 Red 11: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 30 ms y con unas pérdidas del 1% de los paquetes.



Gráfica 41 - Diferencia de tiempos en función del peso para la Red 11

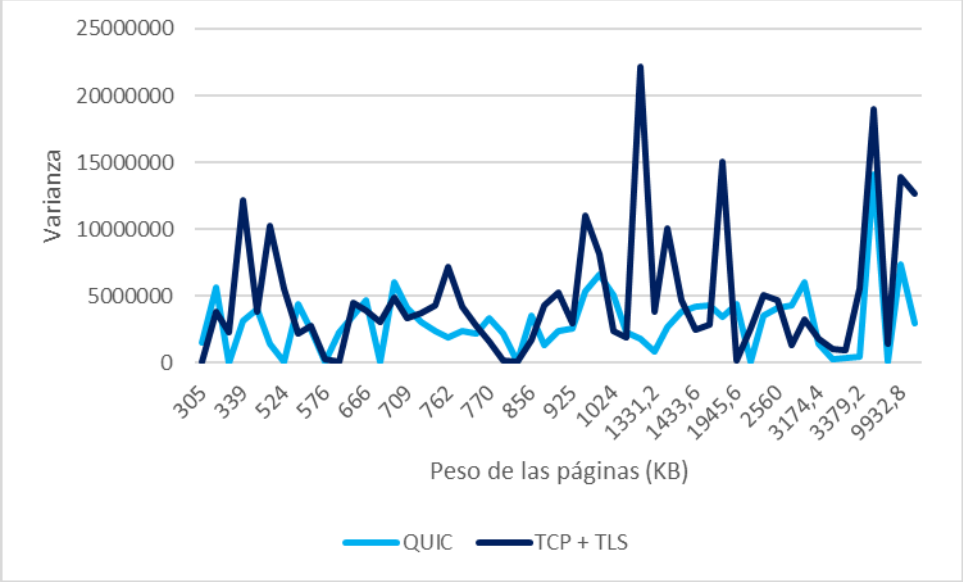
Al igual que en los escenarios previos, el módulo FEC actúa en favor del protocolo QUIC mientras que el ancho de banda también le beneficia, por lo que el predominio del protocolo en este escenario es casi total.

En este escenario, QUIC tiene mejor rendimiento que TCP y TLS para páginas con mayor número de peticiones.

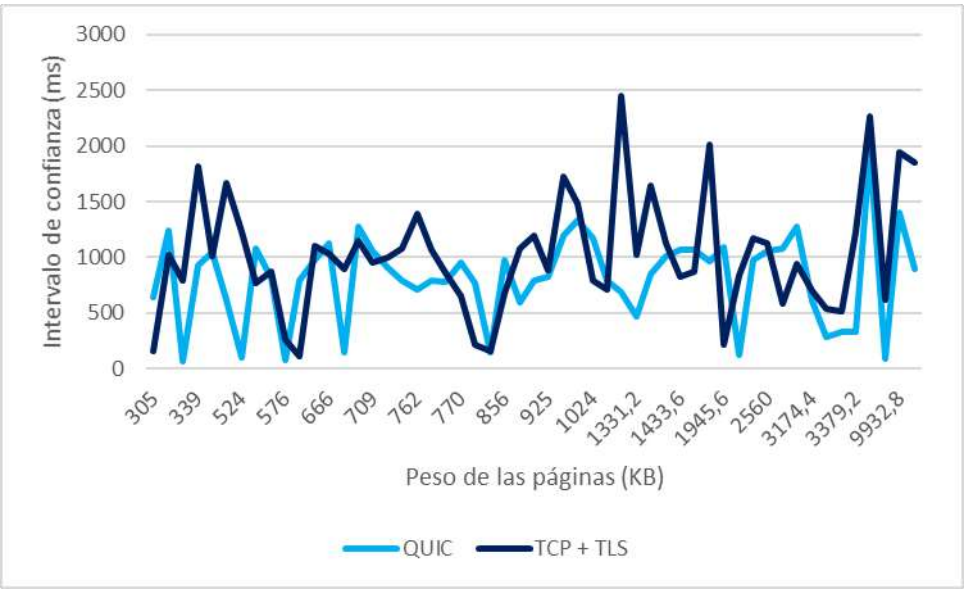


Gráfica 42 - Diferencia de tiempos en función del peso para la Red

Los intervalos de confianza y la varianza siguen estables:

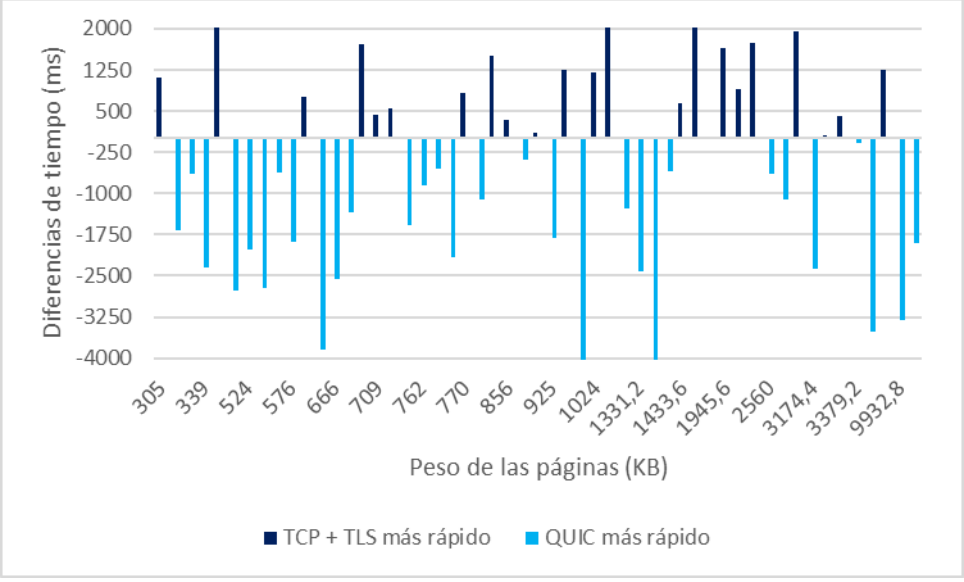


Gráfica 43 - Varianza en la Red 11



Gráfica 44 - Intervalo de confianza en la Red 11

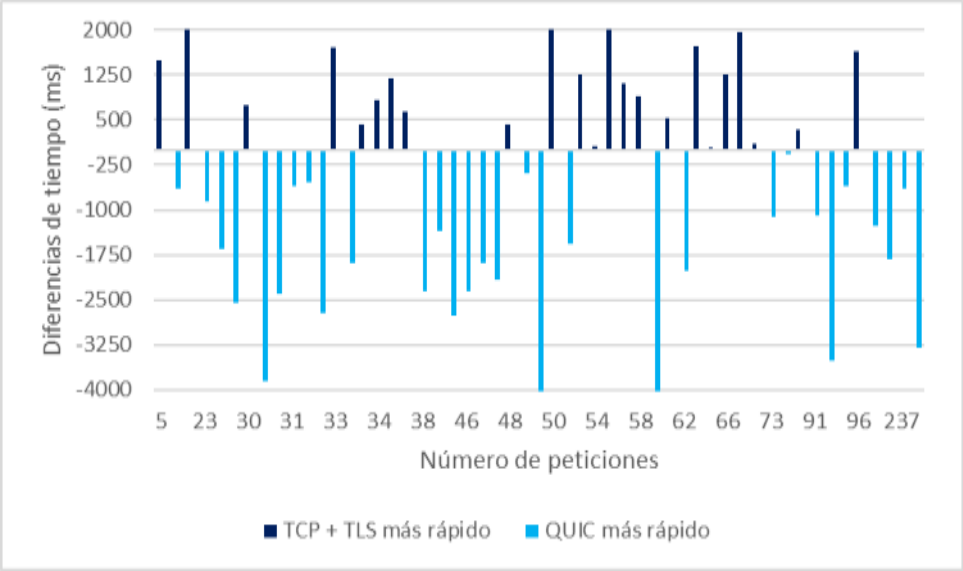
9.2.12 Red 12: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 30 ms y con unas pérdidas del 10% de los paquetes.



Gráfica 45 - Diferencia de tiempos en función del peso para la Red 12

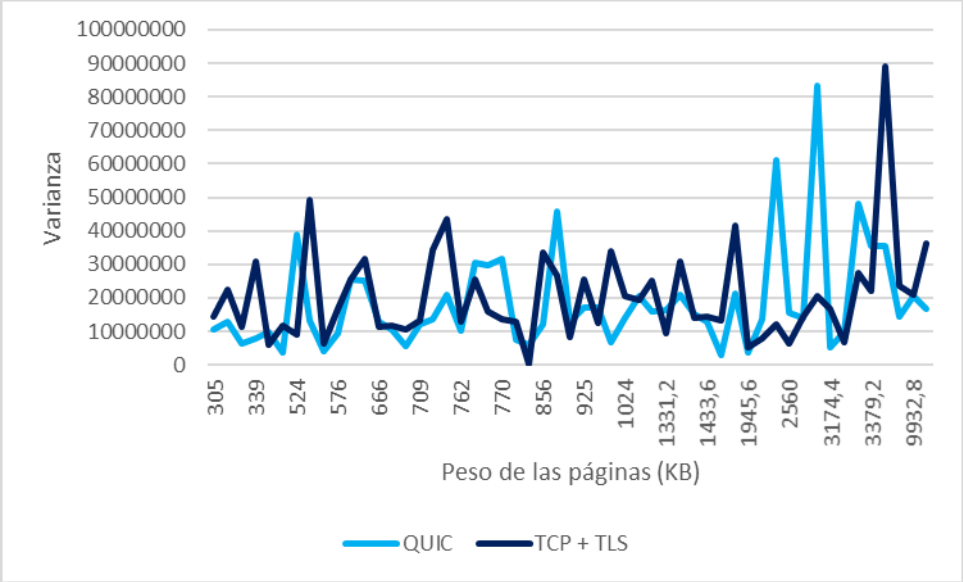
Al igual que previamente, comprobamos que QUIC predomina al ser una red de alta velocidad, aunque su rendimiento se ve afectado al tener una cantidad tan alta de pérdidas, ya que la diferencia de tiempo y el número de páginas en las que QUIC demuestra ser más rápido que TCP y TLS disminuye.

En este escenario el predominio de QUIC también se hace notable, independientemente del número de peticiones de las páginas.

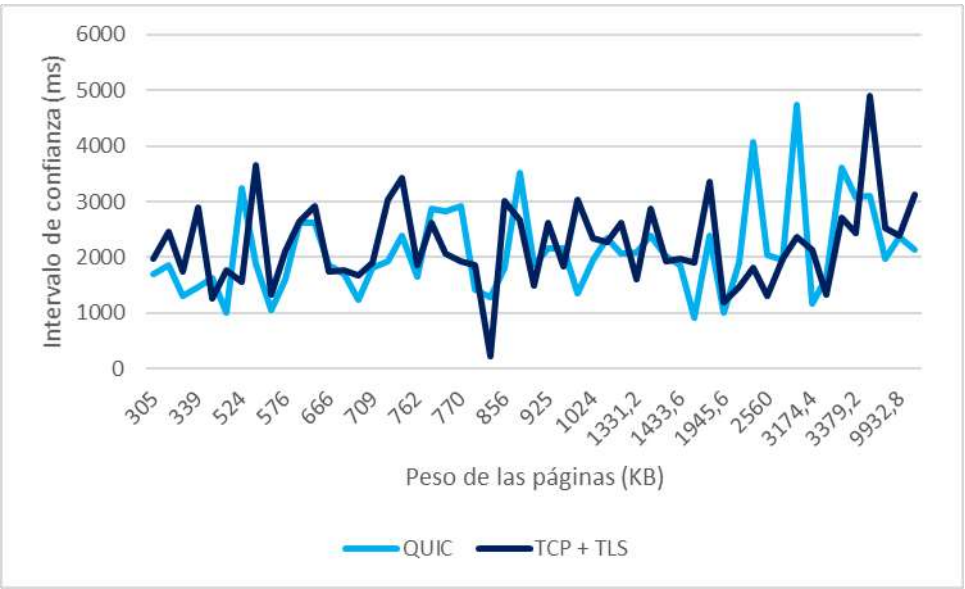


Gráfica 46 - Diferencia de tiempos en función del número de peticiones para la Red 12

El intervalo de confianza y la varianza siguen estables:

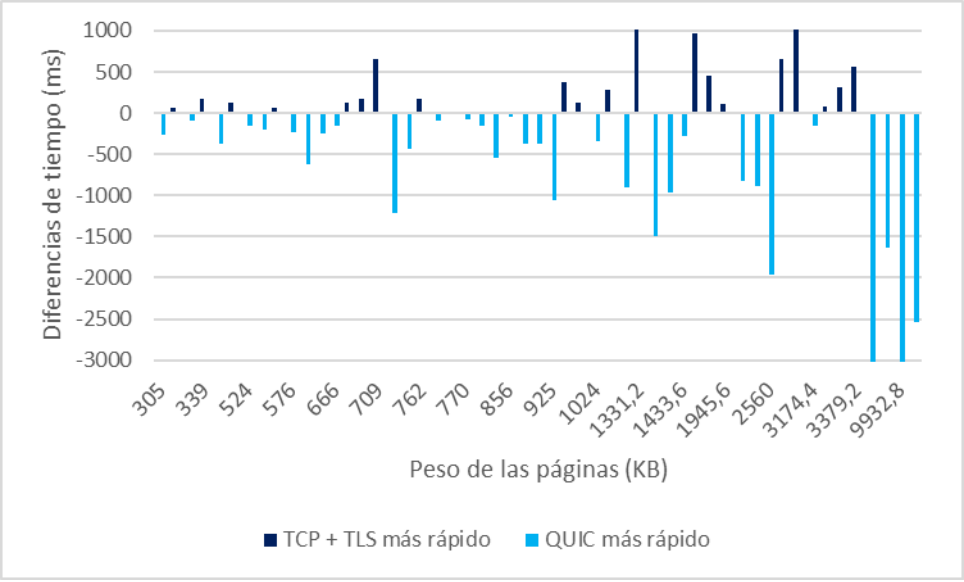


Gráfica 47 - Varianza en la Red 12



Gráfica 48 - Intervalo de confianza en la Red 12

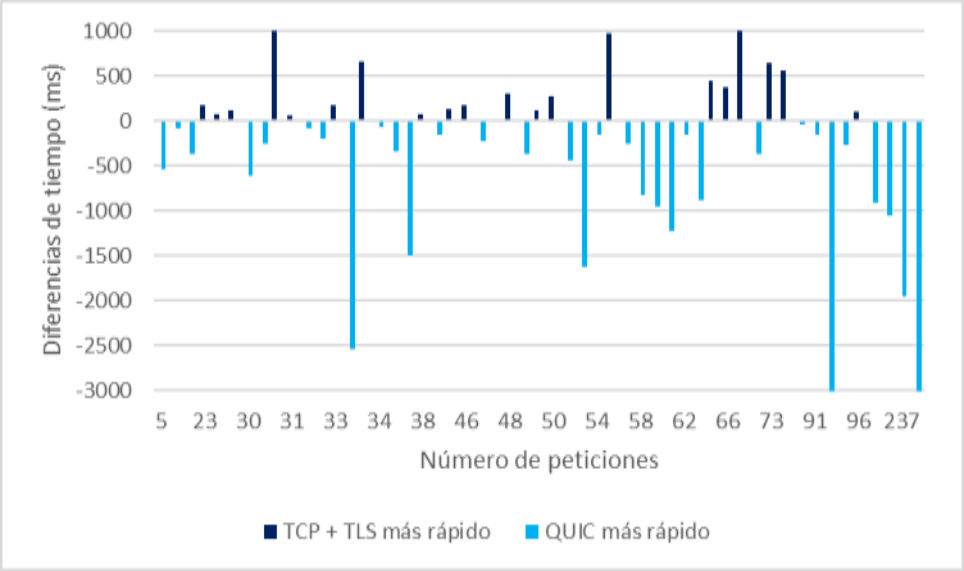
9.2.13 Red 13: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 270 ms y sin pérdidas.



Gráfica 49 - Diferencia de tiempos en función del peso para la Red 13

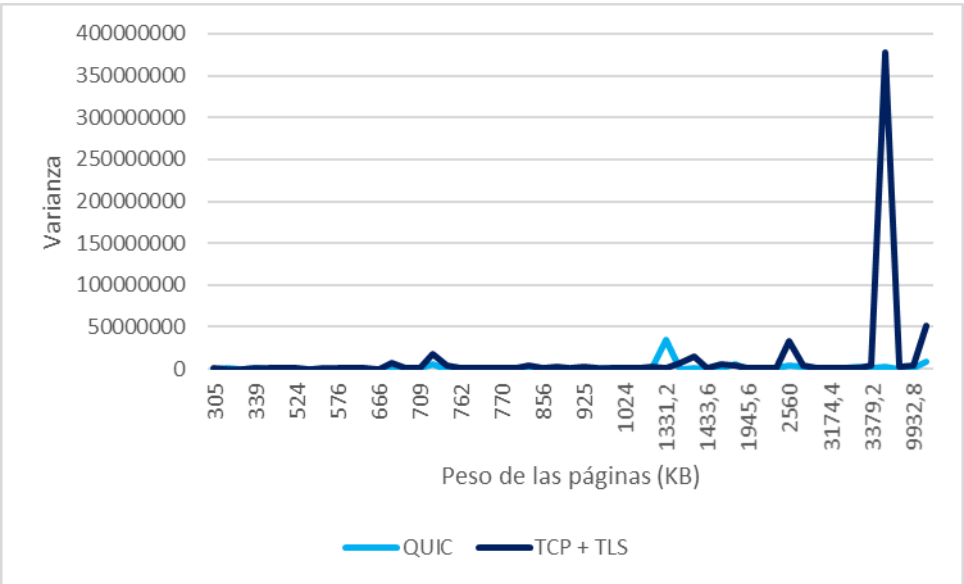
En este escenario podemos observar a simple vista que QUIC mejora en rendimiento casi por completo a los protocolos TCP y TLS, especialmente en las redes de mayor peso donde la diferencia de tiempo se pronuncia.

Al igual que en el caso anterior, en este escenario el predominio de QUIC es generalizado, por lo que no podemos distinguir entre páginas con mayor o menor número de peticiones.

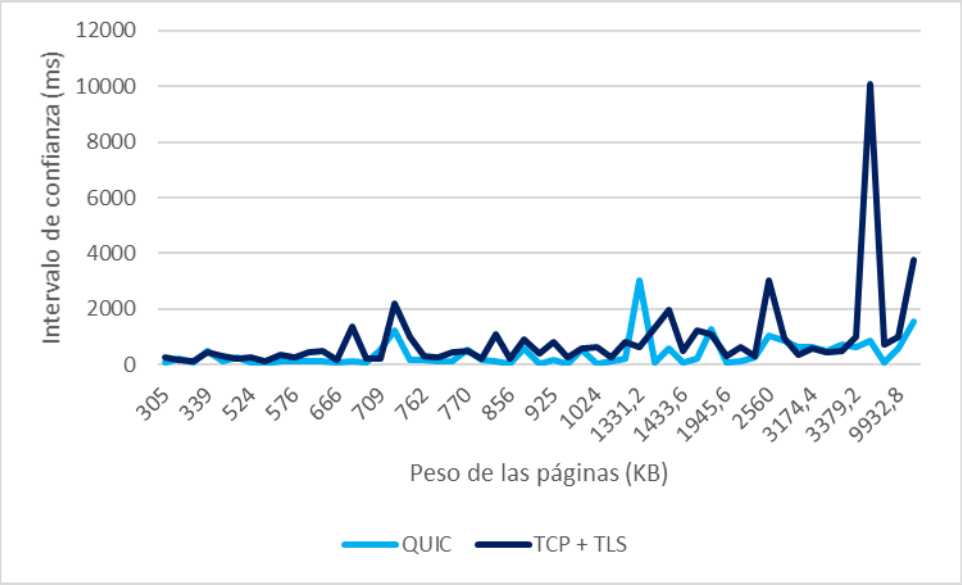


Gráfica 50 - Diferencia de tiempos en función del número de peticiones para la Red 13

Por otro lado, en el intervalo de confianza y la varianza reaparece el pico inicial, por lo que podemos afirmar que, al ejecutarse el script en un bucle, es posible que la bajada en la velocidad de red afectase desde una iteración en este escenario hasta la siguiente iteración en el escenario de la Red 8.

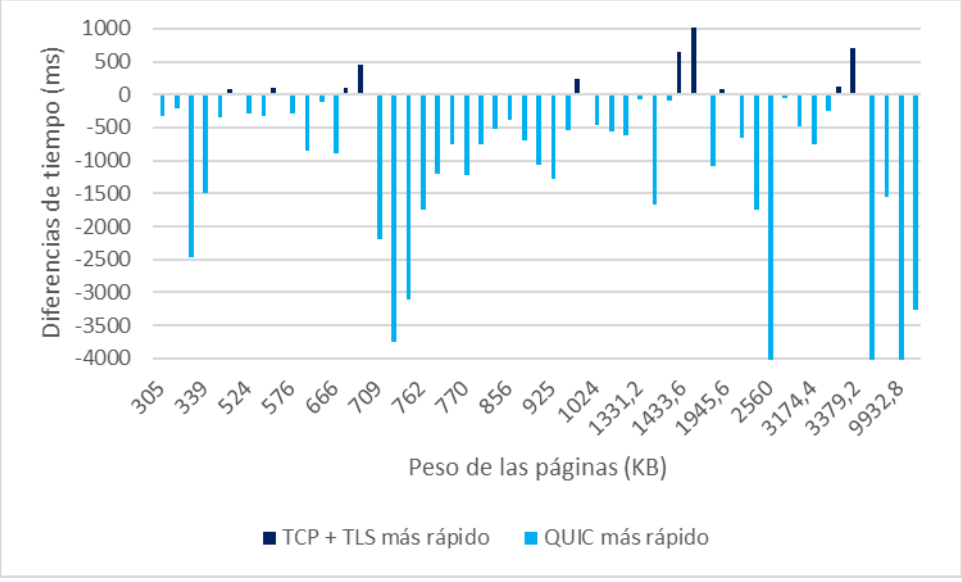


Gráfica 51 - Varianza en la Red 13



Gráfica 52 - Intervalo de confianza en la Red 13

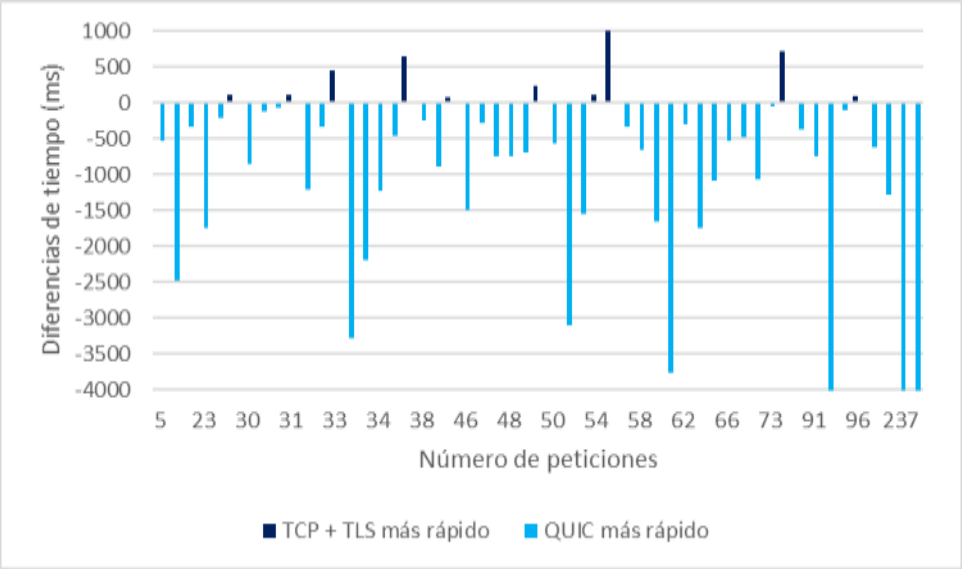
9.2.14 Red 14: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 270 ms y con unas pérdidas del 0,1% de los paquetes.



Gráfica 53 - Diferencia de tiempos en función del peso para la Red 14

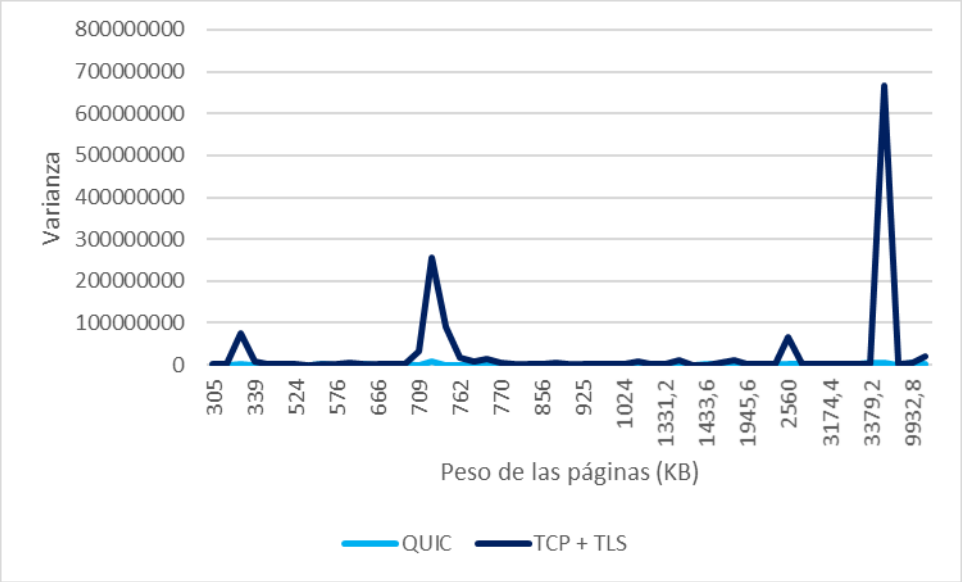
Acabamos de empezar el mejor escenario acorde a la teoría para QUIC, debido a que las redes de alta velocidad, un tiempo de latencia alto y cualquier tipo de pérdidas deberían favorecer a este protocolo. Así lo demostramos en los siguientes escenarios, donde apenas existen páginas que, utilizando los protocolos TCP y TLS, tarden menos que las que utilizan QUIC, y si lo hacen, es con muy poca diferencia.

Dado que en este escenario el predominio de QUIC es prácticamente total, no podemos realizar afirmaciones en cuanto al rendimiento respecto al número de peticiones de las páginas web.

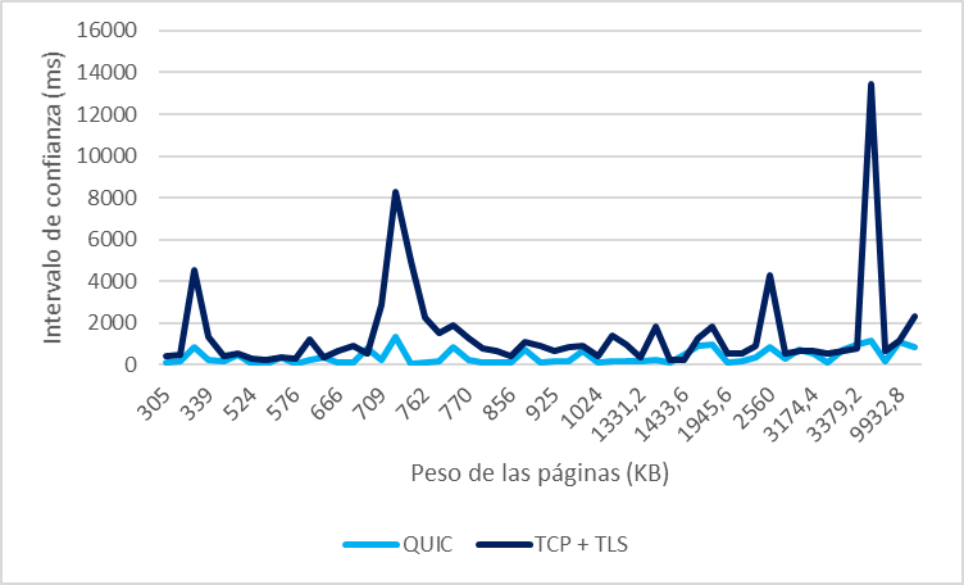


Gráfica 54 - Diferencia de tiempos en función del peso para la Red 14

La varianza e intervalo de confianza muestran una imagen similar a la anterior:

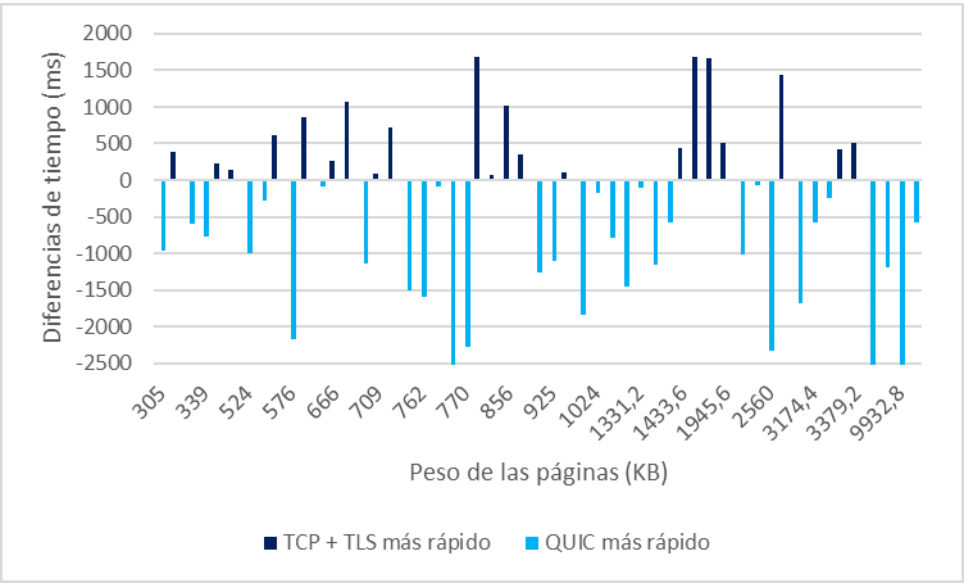


Gráfica 55 - Varianza en la Red 14



Gráfica 56 - Intervalo de confianza en la Red 14

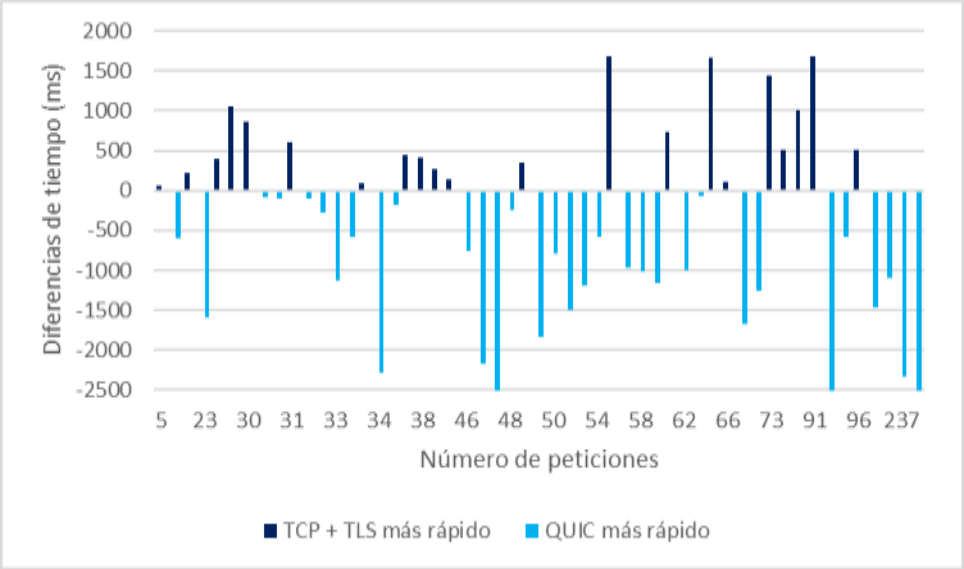
9.2.15 Red 15: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 270 ms y con unas pérdidas del 1% de los paquetes.



Gráfica 57 - Diferencia de tiempos en función del peso para la Red 15

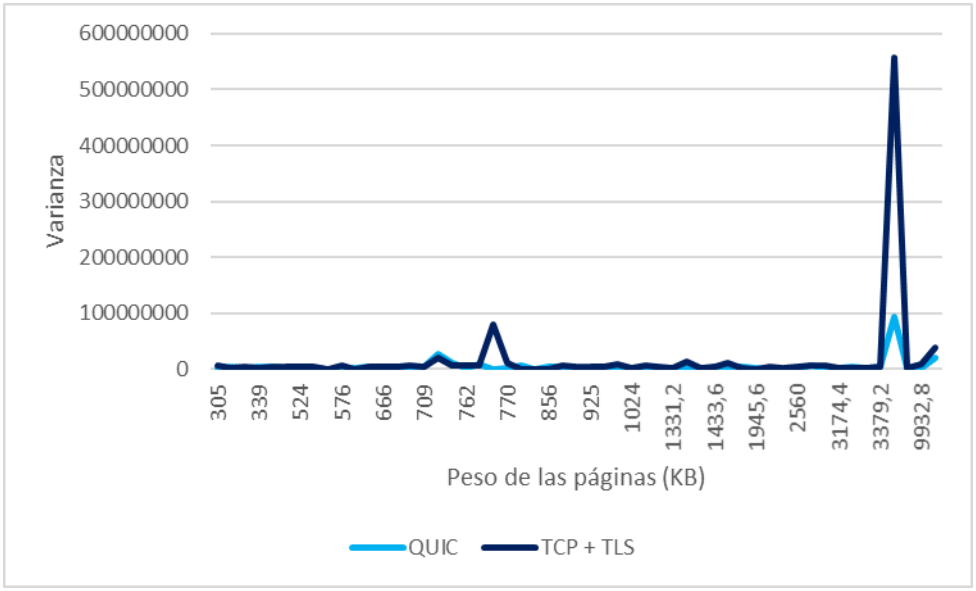
En este escenario, observamos como el rendimiento ha empeorado respecto al escenario anterior (con menos pérdidas), aunque sigue superando a sus protocolos análogos, TCP y TSL.

En este escenario volvemos a observar un mejor rendimiento de QUIC respecto a TCP y TLS para páginas con mayor número de peticiones.

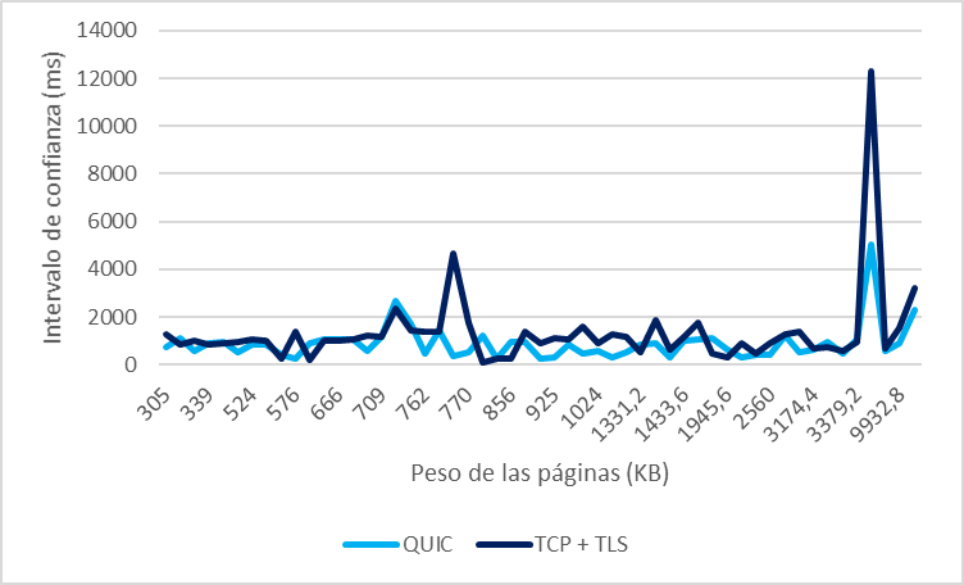


Gráfica 58 - Diferencia de tiempos en función del peso para la Red 15

Observando la varianza y el intervalo de confianza, la situación no ha variado:

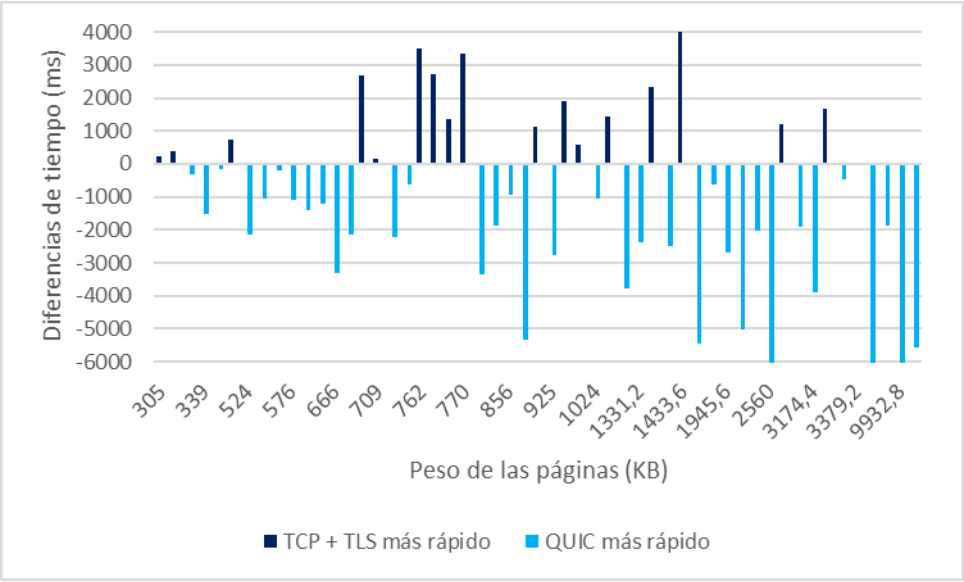


Gráfica 59 - Varianza en la Red 15



Gráfica 60 - Intervalo de confianza en la Red 15

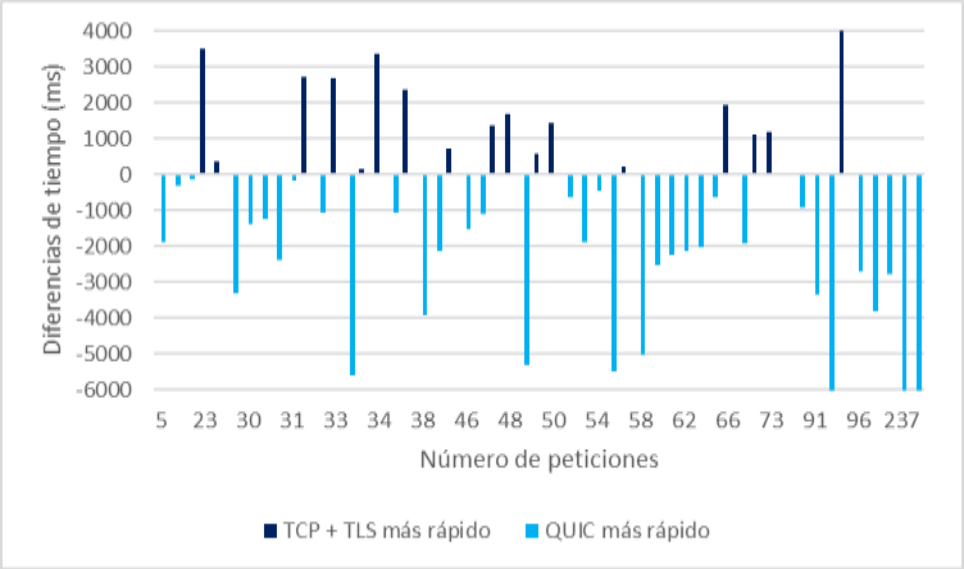
9.2.16 Red 16: Red con una velocidad de subida y bajada de 12 Mb/s, tiempo de latencia de 270 ms y con unas pérdidas del 10% de los paquetes.



Gráfica 61 - Diferencia de tiempos en función del peso para la Red 16

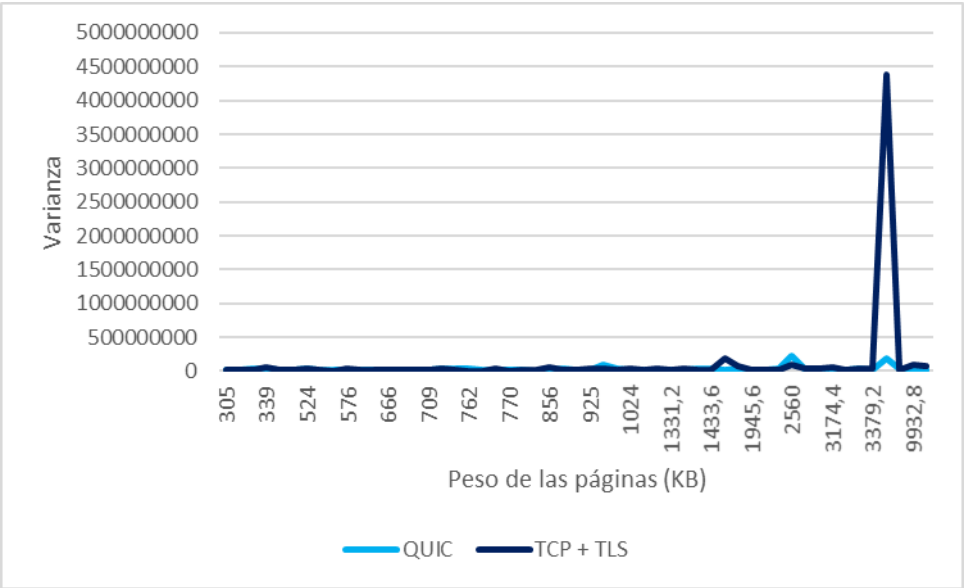
En nuestro último escenario, observamos como QUIC se ve afectado de nuevo por el alto porcentaje de pérdidas, aunque al igual que en los escenarios anteriores, el alto ancho de banda implica que siga teniendo mejor rendimiento que TCP y TLS.

Una vez más, comprobamos un mejor rendimiento de QUIC sobre páginas con un alto número de peticiones:

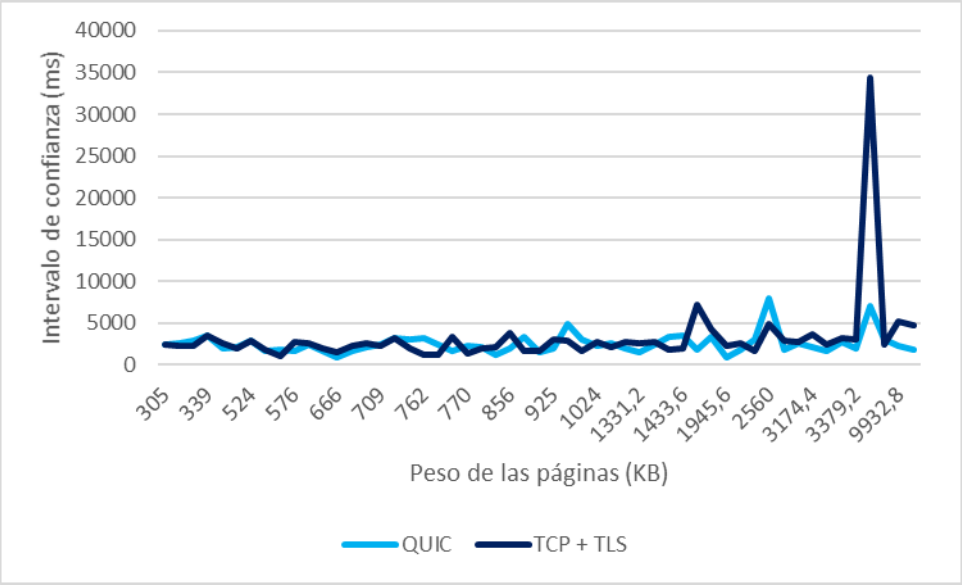


Gráfica 62 - Diferencia de tiempos en función del peso para la Red 16

Observando la varianza y el intervalo de confianza, vemos que en este caso apenas hay picos aparte del ya conocido:



Gráfica 63 - Varianza en la Red 16



Gráfica 64 - Intervalo de confianza en la Red 16

9.3. Resumen de los resultados:

- En redes con un alto ancho de banda, QUIC supera en cuanto a rendimiento a TCP y TLS, aunque su rendimiento se ve afectado negativamente en el caso de escenarios sin pérdidas o en el caso de pérdidas iguales al 10%.
- Por otro lado, en redes con un bajo ancho de banda, no se observa ninguna diferencia apreciable en cuanto al rendimiento general de los protocolos.
- En páginas con un tamaño elevado, QUIC actúa mejor que TCP y TLS.
- El módulo FEC de QUIC es beneficioso sólo en los casos de pérdidas iguales al 0,1% y 1%.
- El aumento de la latencia beneficia al protocolo QUIC.
- Respecto al número de peticiones, QUIC tiene mejor rendimiento sobre escenarios con alta velocidad de red y alta latencia, mientras que TCP y TLS demuestran actuar mejor en los casos contrarios.

10. Conclusiones

10.1 Resumen del proyecto

En este documento hemos analizado uno de los protocolos de nivel aplicación más utilizados hoy en día en Internet, HTTP/2, sobre TCP y TLS como protocolos de transporte, y lo hemos comparado frente a su uso sobre un prometedor protocolo de transporte aún en desarrollo, QUIC, un protocolo de multiplexación sobre UDP, con algunas características similares pero que pretende sustituir a HTTP/2 en un futuro.

Hemos empezado analizando los objetivos de este estudio, así como su marco regulador, el presupuesto del proyecto y el impacto socio-económico que tendría la aplicación de este estudio.

Posteriormente hemos observado las diferentes características del funcionamiento de HTTP/2, hablando sobre el formato de su unidad básica, el *frame*, y hablado sobre el establecimiento de conexión de los dos protocolos de transporte que tiene por debajo por lo general, TCP y su *three-way handshake*, compuesto por un SYN, un SYN-ACK y un ACK, y el establecimiento de conexión de TLS, con los intercambios de mensajes de *Client Hello* y *Server Hello*, el intercambio de claves y certificados, el uso del protocolo *Change Cipher Spec* y los mensajes *Finished* para indicar el fin de la negociación. En total, el establecimiento de conexión de HTTP/2 sobre TCP y TLS varía entre 2 y 3 RTTs, dependiendo de si la conexión TLS se había creado previamente y se puede recuperar o no.

También hemos hablado de las mejoras respecto a sus versiones anteriores, HTTP/1.0 y HTTP/1.1, indicando que es un protocolo binario, que implementa multiplexación que permite priorizar flujos y controlarlos mediante el *frame* `WINDOW_UPDATE`, la compresión de cabeceras HPACK y el servicio "*Server push*" que permite enviar información sin una petición explícita previa por parte del cliente.

Después, hemos hablado de QUIC, del formato de su unidad mínima, el paquete, de cómo permite multiplexación sobre UDP para solventar el problema del bloqueo "*Head of Line*", del establecimiento de conexión con 0-RTTs, que permite disminuir el tiempo de latencia total en comparación con TCP y TLS que requerían entre 2 y 3 RTTs, de cómo implementa la corrección de errores hacia adelante (FEC) para medios con pérdidas. También hemos mencionado los mecanismos de control flexible de congestión y de control de flujo, la prevención de ataques de inyección y manipulación de cabeceras gracias a la autenticación y encriptación del protocolo y de la resistencia a migración de conexiones gracias a su identificador de flujos.

Por último, hemos especificado cómo funciona HTTP/2 sobre QUIC: para especificar su uso como protocolo de transporte, es necesario especificar la cabecera `Alternate-protocol: 443:quic`. Además, relega la administración de flujos a QUIC, pero QUIC utiliza la compresión de cabeceras HPACK lo que puede llevar a bloqueo "*Head of line*" en la transmisión de cabeceras.

Posteriormente, hemos hablado del diseño de la comparativa, en el que hemos realizado diez pruebas sobre cincuenta y tres páginas web, con los dos protocolos distintos (HTTP/2 sobre TCP y TLS y sobre QUIC), bajo dieciséis configuraciones de red distintas (ancho de banda que varía entre 0,6 Mb/s y 12 Mb/s, tiempo de latencia que varía entre 30 ms y 270 ms y pérdidas que van desde un 0% hasta un 10% de los paquetes), sumando un total de 16.960 pruebas.

Para automatizar las medidas del tiempo de carga, hemos utilizado Selenium 2.39.0, Chrome WebDriver 2.32 y Python Virtual Display 0.2.1 en un script en el lenguaje de programación Python, basándonos en el script desarrollado por Somak R. Das para su estudio *"Evaluation of QUIC on Web Page Performance"* del que hablamos en el apartado de "Trabajo Previo". En este script se calcula la diferencia entre el evento `navigationStart` (tiempo en el que el cliente percibe el inicio de carga de una página) y `loadEventEnd` (tiempo en el que sucede el evento Load de una página).

Por otro lado, hemos utilizado otro script sobre el mencionado anteriormente en el que especificamos todos los escenarios de simulación y utilizamos el programa Mahimahi para emular los distintos escenarios de red y realizar la carga de las cincuenta y tres páginas sobre ellos.

Para forzar a Chrome WebDriver a usar el protocolo QUIC, hemos utilizado los flags `--incognito` (para evitar el uso de la memoria caché), `--ignore-certificate-errors` (para asegurarnos que no haya errores que eviten una conexión segura y, por tanto, el uso de HTTP/2), `--enable-quic` (para indicar al navegador que utilice QUIC siempre que sea posible) y `--enable-quic-https` (para indicar al navegador que utilice el encriptado y autenticado propio de QUIC).

Para forzar a Chrome WebDriver a utilizar HTTP/2 sobre TCP y TLS, eliminamos las opciones `--enable-quic` y `--enable-quic-https` y habilitamos la opción `--disable-quic` para asegurarnos de que usará TCP y TLS en lugar de QUIC.

Tras analizar los resultados obtenidos, hemos llegado a las siguientes conclusiones:

- En redes con alto ancho de banda, QUIC supera a TCP y TSL en todos los escenarios probados, aunque su rendimiento se ve afectado negativamente en los casos sin pérdidas o de pérdidas iguales a un 10%.

- En redes con bajo ancho de banda, no se observa ninguna diferencia apreciable en el rendimiento de ambos protocolos.
- Por lo general, QUIC tiene mejor rendimiento sobre páginas de gran tamaño.
- El módulo FEC que implementa QUIC beneficia a dicho protocolo para pérdidas comprendidas entre el 0,1% y 1%. En casos sin pérdidas el módulo FEC no se aplica, mientras que en casos con muchas pérdidas TCP y TLS suelen tener mejor rendimiento que bajo las mismas condiciones con menos pérdidas.
- El aumento de latencia beneficia al protocolo QUIC, aumentando su rendimiento.
- En redes con bajo ancho de banda y baja latencia, TCP y TSL actúan mejor sobre páginas con mayor número de peticiones, mientras que en páginas con alto ancho de banda y latencia, QUIC supera a TCP y TLS.

En conclusión, teniendo en cuenta los resultados de estudios anteriores, observamos que los años de desarrollo del protocolo QUIC empiezan a dar sus frutos, obteniendo mejores resultados que HTTP/2 sobre TCP y TSL en muchas ocasiones. Pese a eso, aún queda mucho trabajo por delante en relación con este protocolo para poder obtener el alcance que hoy en día tiene HTTP/2 y convertirse realmente en el futuro de los protocolos para páginas web.

10.2. Problemas y limitaciones del estudio

10.2.1 Problemas surgidos.

Debido a que QUIC es un protocolo que aún está en desarrollo, existe información limitada al respecto y es complicado definir unas líneas de trabajo desde un punto inicial. Esto ha causado muchos retrasos respecto a la planificación inicial debido a que había que probar los distintos medios para forzar QUIC hasta dar con el más adecuado.

Por otro lado, existían herramientas que facilitaban el estudio del protocolo QUIC sobre páginas de cualquier servidor (no sólo los de Google Inc.), como QuicShell, desarrollada por el Massachusetts Institute of Technology, que al inicio del estudio se encontraban disponibles en internet pero cuando llegó la hora de utilizarla se había retirado de su repositorio en GitHub y no fue posible encontrarla, por lo que hubo que reducir el alcance de este proyecto a páginas pertenecientes a Google Inc.

10.2.2. Limitaciones del estudio.

Como comentamos, dado que el protocolo QUIC está en desarrollo por Google Inc., actualmente sólo los servidores pertenecientes a dicha empresa comprenden QUIC y son capaces de utilizar ese protocolo para cargar sus páginas web, al contrario que

HTTP/2 sobre TCP y TLS. Esto limita mucho la cantidad de páginas que están disponibles para realizar las pruebas.

Por falta de tiempo del proyecto, ya que se perdió mucho tiempo encontrando la forma correcta de utilizar el protocolo QUIC, hemos dejado escenarios sin probar que podrían haber sido interesantes (ver 10.3 Trabajos futuros).

Por otro lado, debido a que no disponemos de una conexión a internet aislada, nos hemos visto obligados a realizar las pruebas bajo una conexión por cable casera, por lo que los resultados se han visto afectados por picos de actividad en la red que han podido introducir alguna distorsión.

Por otro lado, cuatro días antes de la entrega de esta memoria se ha publicado una nueva versión de QUIC con bastantes cambios en comparación con la evaluada en este documento. Debido a la falta de tiempo para implementar los cambios, hemos decidido dejar este estudio en base a la versión anterior e incluir como trabajo futuro la repetición del estudio bajo esta nueva versión.

10.3 Trabajos futuros

Debido al tiempo limitado del estudio, o a la disponibilidad de herramientas o funciones de QUIC, no hemos podido realizar los siguientes estudios, aunque en un futuro sería interesante comprobar el rendimiento de ambos protocolos bajo las siguientes condiciones:

- Evaluar el comportamiento de ambos protocolos bajo redes celulares: Actualmente hemos comprobado el comportamiento sólo bajo redes Ethernet, pero hemos dejado fuera del alcance del estudio las redes celulares. Sería interesante comprobar si el rendimiento de HTTP/2 sobre QUIC cambia bajo estas condiciones de red y en qué medida lo hace.
- Comprobar con páginas que no pertenezcan a Google Inc.: Como hemos explicado previamente, debido a la falta de herramientas hemos sido capaces de evaluar el rendimiento de QUIC únicamente sobre páginas alojadas en servidores pertenecientes a dicha empresa. En un futuro, se debería completar el estudio realizando las pruebas sobre un rango de páginas web más amplio, incluyendo aquellas alojadas en otros servidores.
- Comprobar el funcionamiento una vez esté disponible TLS v1.3, debido a que cambiará el establecimiento de conexión de QUIC y probablemente se asemeje más al utilizado por TCP + TLS.
- Realizar el estudio bajo la nueva versión de QUIC para observar si existen cambios significativos respecto a los resultados del documento.

- Realizar el estudio bajo una red más estable, que no sufra de picos de actividad que pueda afectar a las medidas de los protocolos.

11. Bibliografía

[1] Leiner, B., Cerf, V., Clark, D., Kahn, R., Kleinrock, L., & Lynch, D. et al. (1997). Brief History of the Internet | Internet Society. Internet Society. Retrieved 24 September 2017, from https://www.internetsociety.org/internet/history-internet/brief-history-internet/

[2] Kim, B. (2005). Internationalizing the Internet (p. 53). Cheltenham, UK: Edward Elgar Pub.

[3] QUIC at 10,000 feet. Google Docs. Retrieved 24 September 2017, from https://docs.google.com/document/d/1gY9-YNDNAB1eip-RTPbqphgySwSNSDHLq9D5Bty4FSU/edit

[4] Belshe, M., Peon, R., & Thomson, Ed, M. (2015). RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2). Tools.ietf.org. Retrieved 5 September 2017, from https://tools.ietf.org/html/rfc7540

[5] Hamilton, R., Iyengar, J., Swett, I., & Wilk, A. (2016). QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. Tools.ietf.org. Retrieved 5 September 2017, from https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02

[6] Ravinet. (2014). Ravinet selenium script to automate page loads. GitHub. Retrieved 24 September 2017, from https://github.com/ravinet/sites/blob/scripts/load_page.py

[7] Somak R. Das. (2014). Evaluation of QUIC on Web Page Performance (Master Thesis). Massachusetts Institute of Technology.

[8] Netravali, R., Sivaraman, A., Hill, G., & Winstein, K. Mahimahi. Mahimahi.mit.edu. Retrieved 24 September 2017, from http://mahimahi.mit.edu

[9] MARCOM, S. (2016). Sandvine - Global Internet Phenomena. Sandvine.com. Retrieved 24 September 2017, from https://www.sandvine.com/trends/global-internet-phenomena/

[10] HTTP/2 Frequently Asked Questions. Http2.github.io. Retrieved 5 September 2017, from https://http2.github.io/faq

[11] About the IETF. Ietf.org. Retrieved 24 September 2017, from https://www.ietf.org/about/

[12] Mission Statement. Ietf.org. Retrieved 24 September 2017, from https://www.ietf.org/about/mission.html

[13] RFC 2026 - The Internet Standards Process -- Revision 3. (2013). Datatracker.ietf.org. Retrieved 24 September 2017, from https://datatracker.ietf.org/doc/rfc2026/

[14] Marco regulador de las comunicaciones electrónicas. (2015). Eur-lex.europa.eu. Retrieved 24 September 2017, from http://eur-lex.europa.eu/legal-content/ES/TXT/?uri=LEGISSUM:l24216a

[15] Maugard, J. (2015). CNMC, el regulador de las telecomunicaciones en España. KillMyBill Espagne. Retrieved 24 September 2017, from https://www.killmybill.es/cnmc-regulador-telecomunicaciones/

[16] CNMC | Comisión Nacional de los Mercados y la Competencia. Ámbitos de actuación: Telecomunicaciones. Cnmc.es. Retrieved 24 September 2017, from https://www.cnmc.es/ambitos-de-actuacion/telecomunicaciones

[17] HTTP/2 Last Call! – Webtide. (2014). Webtide.com. Retrieved 5 September 2017, from https://webtide.com/http2-last-call/

[18] M. Belshe, Twist, R. Peon, & Google, Inc. (2012). draft-mbelshe-httpbis-spdy-00 - SPDY Protocol. Tools.ietf.org. Retrieved 24 September 2017, from https://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00

[19] HTTP/2 - High Performance Browser Networking (O'Reilly). High Performance Browser Networking. Retrieved 5 September 2017, from https://hpbn.co/http2/

[20] T. Berners-Lee, MIT/LCS, R. Fielding, UC Irvine, & H. Frystyk. (1996). RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0. Tools.ietf.org. Retrieved 24 September 2017, from https://tools.ietf.org/html/rfc1945

[21] R. Fielding, UC Irvine, J. Gettys, Compaq/W3C, J. Mogul, & Compaq et al. (1999). RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1. Tools.ietf.org. Retrieved 24 September 2017, from https://tools.ietf.org/html/rfc2616

[22] Information Sciences Institute (University of Southern California). (1981). RFC 793 - Transmission Control Protocol. Tools.ietf.org. Retrieved 24 September 2017, from https://tools.ietf.org/html/rfc793

[23] T. Dierks, E. Rescorla, & RTFM, Inc. (2008). RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2. Tools.ietf.org. Retrieved 24 September 2017, from https://tools.ietf.org/html/rfc5246

[24] Lin, Z. (2015). TLS Session Resumption: Full-speed and Secure. Cloudflare Blog. Retrieved 24 September 2017, from https://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/

[25] Garza, J. (2017). How does HTTP/2 solve the Head of Line blocking... | Akamai Community. Community.akamai.com. Retrieved 24 September 2017, from https://community.akamai.com/community/web-performance/blog/2017/08/10/how-does-http2-solve-the-head-of-line-blocking-hol-issue

[26] Douglas, S. An in-depth overview of HTTP/2 · JBoss Community. Undertow.io. Retrieved 5 September 2017, from http://undertow.io/blog/2015/04/27/An-in-depth-overview-of-HTTP2.html

[27] R. Peon, Google, Inc, H. Ruellan, & Canon CRF. (2015). RFC 7541 - HPACK: Header Compression for HTTP/2. Tools.ietf.org. Retrieved 24 September 2017, from https://tools.ietf.org/html/rfc7541

[28] Delabassee, D. (2015). HTTP/2 and Server Push. Blogs.oracle.com. Retrieved 24 September 2017, from https://blogs.oracle.com/theaquarium/http2-and-server-push

[29] QUIC Geek FAQ (for folks that know about UDP, TCP, SPDY, and stuff like that). Department of Computer Science and Engineering | Indian Institute of Technology Bombay. Retrieved 24 September 2017, from https://www.cse.iitb.ac.in/~cs641/references/05-quic.pdf

[30] E. Rescorla, & RTFM, Inc. (2017). draft-ietf-tls-tls13-21 - The Transport Layer Security (TLS) Protocol Version 1.3. Tools.ietf.org. Retrieved 24 September 2017, from https://tools.ietf.org/html/draft-ietf-tls-tls13-21

[31] Langley, A., & Chang, W. (2016). QUIC Crypto. Google Docs. Retrieved 24 September 2017, from https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit

[32] Swett, I. (2016). QUIC FEC v1. Google Docs. Retrieved 24 September 2017, from https://docs.google.com/document/d/1Hg1SaLEl6T4rEU9j-isovCo8VEjjnuCPTcLNJewj7Nk/edit

[33] Shade, R. (2016). Flow control in QUIC. Google Docs. Retrieved 24 September 2017, from https://docs.google.com/document/d/1F2YfdDXKpy20WVKJueEf4abn_LVZHhMUMS5gX6Pgjl4/edit#

[34] Carlucci, G., De Cicco, L., & Mascolo, S. (2015). HTTP over UDP: an experimental investigation of QUIC. Lecture, ACM Symposium on Applied Computing. Salamanca, Spain.

[35] Megyesi, P., Krämer, Z., & Molnár, S. (2016). How quick is QUIC. Lecture, IEEE ICC 2016 - Communication QoS, Reliability and Modeling Symposium.

[36] Selenium 2.39.0: Python Package Index. Pypi.python.org. Retrieved 24 September 2017, from https://pypi.python.org/pypi/selenium/2.39.0

[37] ChromeDriver - WebDriver for Chrome. Sites.google.com. Retrieved 24 September 2017, from https://sites.google.com/a/chromium.org/chromedriver/getting-started

[38] PyVirtualDisplay 0.2.1 : Python Package Index. Pypi.python.org. Retrieved 24 September 2017, from https://pypi.python.org/pypi/PyVirtualDisplay

[39] Software de hojas de cálculo Microsoft Excel 2016. Products.office.com. Retrieved 24 September 2017, from https://products.office.com/es-es/excel

Anexo I - Espectro de páginas web utilizado para el estudio.

1	https://www.google.es/intl/es_es/sky/
2	https://books.google.com/ngrams
3	https://www.google.com/publicdata/directory
4	https://adwords.google.com/home/tools/display-planner/
5	https://www.google.com
6	https://www.thinkwithgoogle.com/intl/es-419/
7	https://www.google.es/services/#?modal_active=none
8	https://fonts.google.com/
9	https://www.google.es/shopping?hl=es_ES
10	https://www.google.com/mars/
11	https://voice.google.com/about
12	https://www.google.es/intl/es/grants/
13	https://www.google.com/trends/correlate
14	https://www.google.es/alerts
15	https://cse.google.es/cse/
16	https://golang.org/
17	https://cloud.google.com/
18	https://www.google.com/culturalinstitute/beta/?hl=es
19	https://developers.google.com/speed/pagespeed/
20	https://trends.google.es/trends/
21	https://domains.google/#/
22	https://patents.google.com/?q=cats
23	https://www.google.com/moon/
24	https://www.google.com/trends/correlate/comic
25	https://www.google.es/admob/
26	https://www.google.com/finance
27	https://maps.google.com/landing/transit/index.html
28	https://news.google.com
29	https://www.google.es/intl/es/docs/about/
30	https://www.google.es/partners/about/index.html
31	https://www.google.es/intl/es/streetview/
32	https://www.google.com/intl/en/drive/
33	https://archive.google.com/intl/es/press/zeitgeist2010/
34	https://play.google.com/music/listen?authuser

35 https://developers.google.com/
36 https://www.tensorflow.org/
37 https://www.google.es/flights/
38 https://www.youtube.com
39 https://sites.google.com/site/fusiontablestalks/stories
40 https://www.google.es/maps
41 https://www.blogger.com/about/?r=1-null_user
42 https://www.google.es/inbox/
43 https://plus.google.com/collections/featured?hl=es
44 https://edu.google.com/intl/es-419/products/productivity-tools/classroom/
45 https://www.google.es/adsense/start/#/?modal_active=none
46 https://www.google.com/recaptcha/intro/invisible.html
47 https://www.google.com/analytics/surveys/#?modal_active=none
48 https://www.doubleclickbygoogle.com/es/
49 https://www.google.com/intl/es419/insidesearch/features/search/knowledge.html
50 https://3dwarehouse.sketchup.com/?hl=es
51 https://www.google.com/gmail/about/
52 https://santatracker.google.com/intl/es-419/village.html
53 https://hangouts.google.com/?hl=es

Anexo II - Diagrama de Gantt del proyecto.

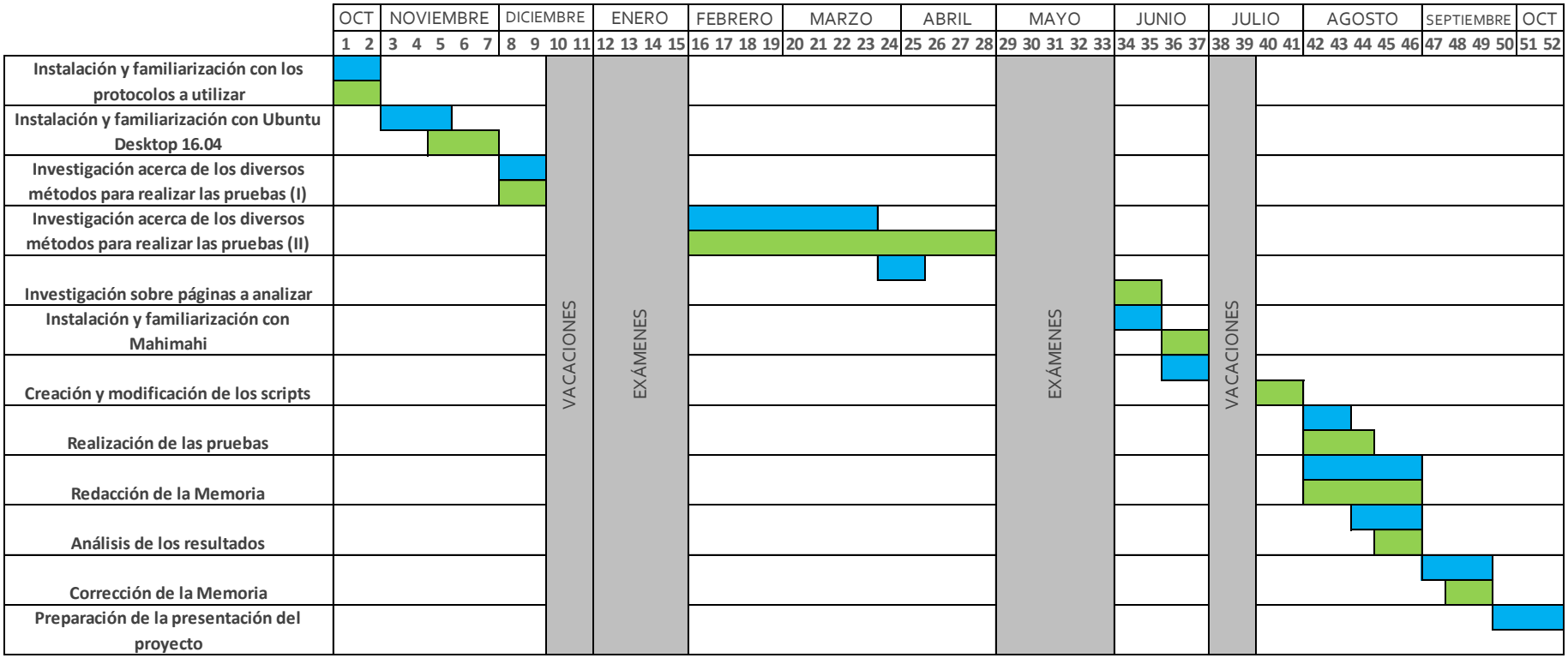


Figura 15 - Diagrama de Gantt del proyecto

Anexo III- Script modificado load_page.py.

```
import sys
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.keys import Keys
from pyvirtualdisplay import Display
import os
from time import sleep

site = sys.argv[1]
delay = int(sys.argv[2])*2
link_speed = sys.argv[3]
loss = sys.argv[4]
if sys.argv[5] == "replay":
    file_name = "replay_summary_" + str(link_speed) + "Mbps_" +
    str(delay) + "ms_" + str(loss) + "perc_loss.txt"
    results = open( file_name, 'a' )

### to not display the page in browser ###
display = Display(visible=0, size=(800,600))
display.start()

### to run chrome ###
options=Options()
options.add_argument("--incognito")
options.add_argument("--ignore-certificate-errors")
options.add_argument("--enable-quic") #Only to run QUIC
options.add_argument("--enable-quic-https") #Only to run QUIC
options.add_argument("--disable-quic") #Only to run TCP + TLS
chromedriver = "/home/lbalart/Descargas/chromedriver"
driver=webdriver.Chrome(chromedriver, chrome_options=options)

driver.set_page_load_timeout(500)
driver.get(site)

#beginning of page load as perceived by user (same as fetchstart
if no previous document)
navigationStart = driver.execute_script("return
window.performance.timing.navigationStart")

#end of page load
loadEventEnd = driver.execute_script("return
window.performance.timing.loadEventEnd")

#total time taken to send a request to the server and receive
the response: responseEnd - requestStart
```

```
while loadEventEnd == 0:
    loadEventEnd = driver.execute_script("return
window.performance.timing.loadEventEnd")
complete_process = loadEventEnd - navigationStart

print "Entire process (navigation and page load): %s" %
complete_process
if sys.argv[5] == "replay":
    results.write( site + " "  + str(complete_process) + "\n")

driver.quit()
display.stop()
```

Anexo IV - Script pruebas.py.

```
import os
import sys
from time import sleep

link_speeds = [0.6, 12]
delays = [rtt / 2 for rtt in [30, 270]]
losses = [0, 0.001, 0.01, 0.1]

site_list = sys.argv[1]

i=1
while i<11:
  for link_speed in link_speeds:
    for delay in delays:
      for loss in losses:
        print "link_speed =", link_speed, ", delay =", delay, ",
loss =", loss, ", iteration=", i

        with open(site_list) as f:
          file_name = "replay_summary_" + str(link_speed) + "Mbps_"
          + str(delay*2) + "ms_" + str(loss) + "perc_loss.txt"

          with open( file_name, 'a' ) as rec:
            rec.write("------------"+str(i)+"-------------\n")
            rec.close()
          for line in f:
            site = line.split( " " )[0].strip("\n")
            print site
            os.system( "mm-loss uplink "+str(loss) + " mm-loss
            downlink " + str(loss) + " mm-delay " + str(delay)
            + " mm-link " +str(link_speed) + "Mbps_trace.txt "
            + str(link_speed) + "Mbps_trace.txt" + "
            /usr/bin/python
            /home/lbalart/Descargas/MEDIDAS_HTTP/load_page_http
            .py " + site + " " + str(delay) + " " +
            str(link_speed) + " " + str(loss) +" replay " )
          f.close()
   i=i+1
```